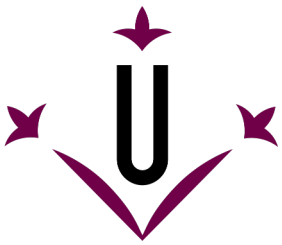

Universitat de Lleida
Escola Politècnica Superior

Enginyeria Tècnica en Informàtica de Sistemes

Treball de final de carrera

Implementació dels codis de Golay en el paquet matemàtic SAGE

Autor:
Gerard Bosch Monserrate

Director:
Dr. Ramiro Moreno Chiral

12 de gener de 2010

A la meva mare,
per donar-me suport i confiar en mi.

Agraïments

Abans de res voldria donar les gràcies a Ramiro, qui fou l'instigador de la idea per al desenvolupament d'aquest treball i qui s'ha mostrat receptiu amb mi des del primer moment davant la idea de la realització d'un treball de teoria de codis. Li he d'agrair tant les tutories i la supervisió que m'ha realitzat, com el material bibliogràfic que m'ha facilitat així com la tasca de recerca que ha realitzat pel seu compte. També li dec els coneixements que he adquirit durant aquest període.

En segon lloc agrair a la gent de #sage-devel del IRC freenode i de la mateixa llista de correu sage-devel, per resoldre alguns petits dubtes o problemes sorgits en la implementació i en el procés de desenvolupament que segueix Sage.

Preàmbul

La motivació del present treball va sorgir de la necessitat de descodificar els codis de Golay —un tipus de codis lineals perfectes— en el paquet matemàtic Sage. Sage és un paquet software de lliure distribució i en actual desenvolupament i popularització destinat a aglutinar les funcionalitats de part dels paquets d'anàlisi matemàtic, calculadors simbòlics i manipuladors algebraics propietaris com Mathematica, Matlab, Maple, Magma,...

En el document es descriurà la implementació realitzada, destacant-ne els aspectes més rellevants. Per a tal efecte, es donarà una introducció als codis lineals i als seus aspectes matemàtics tant des de la vessant de la definició com de les propietats; i al paquet Sage.

A continuació es descriu breument el contingut de cadascun dels capítols que conformen aquest treball.

El capítol 1 presenta les idees generals entorn a un procés de comunicació digital i als codis correctors d'errors. Es presenten també de forma introductòria els codis de Golay i el paquet matemàtic Sage.

El capítol 2 dóna una sèrie de conceptes matemàtics previs referents als codis en general i necessaris abans d'introduir els codis lineals.

El capítol 3 presenta de manera formal els codis lineals i els processos de codificació i descodificació sobre aquests.

El capítol 4 presenta formalment els anomenats codis perfectes i els codis de Golay —vèrtex sobre el qual gira aquest treball—, per als que se'n veuen les propietats, la construcció i la descodificació.

El capítol 5 dóna una visió general sobre el paquet de programari matemàtic Sage i pretén introduir al lector en els aspectes i característiques bàsics.

El capítol 6 tracta sobre la implementació realitzada dels codis binaris de Golay en el paquet Sage. En aquest capítol, es descriu l'estat de Sage en el marc de la teoria de codis i es descriuen també les noves característiques implementades. Finalment es presenta un conjunt de proves de rendiment on es contrasten les millores.

Índex

Índex de figures

Índex de taules

Índex d'algorismes

Capítol 1

Introducció

Es pot definir de manera informal un codi corrector com una tècnica de codificació basada en la redundància. Això consisteix en afegir informació addicional a un missatge, de manera que se n'augmenta la llargada a costa de dotar aquest missatge d'una certa capacitat per corregir i/o detectar errors que pugui contenir.

En el procés de comunicació, la informació es transmet a través d'un canal de comunicació i habitualment aquest canal no és completament fiable. Considerarem el canal com un canal simètric binari teòric on la probabilitat d'error és p i la probabilitat de no–error per tant és $1 - p$:

- El canal és binari, donat que transmet uns i zeros;
- no té pèrdua d'informació, és a dir a la sortida del canal es reben el mateix nombre de símbols que s'han enviat;
- està sotmès a soroll, la naturalesa del qual pot ser diversa;
- és simètric, és a dir, la probabilitat d'intercanviar un 0 per un 1 és la mateixa que la d'intercanviar un 1 per un 0;
- el fet que el canal intercanvii un símbol, és independent a que intercanvii el següent o qualsevol altre.

A més a més,

- la probabilitat d'error p (on $0 \leq p \leq 1$) defineix el canal i generalment aquesta probabilitat és petita. De fet si la probabilitat és $p = 1/2$, direm que el canal és completament aleatori i qualsevol esforç per corregir errors és inútil;
- els errors es distribueixen aleatòriament en el missatge segons la probabilitat p. Es pot veure el canal com un dispositiu que llegeix símbols i els intercanvia amb probabilitat p.

La figura 1.1 mostra de forma esquemàtica aquest canal.

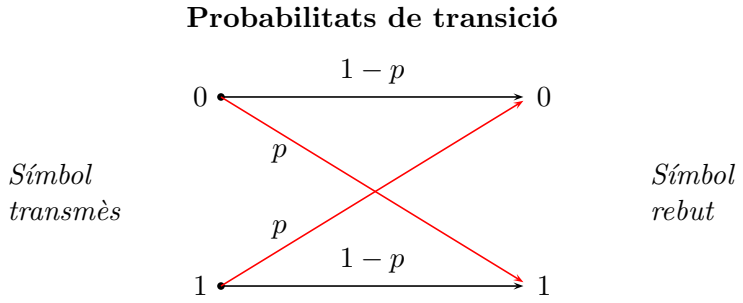

Figura 1.1: Esquema d'un canal simètric binari (BSC).

La codificació d'un missatge està destinada a corregir possibles errors en un procés de transmissió d'informació a través d'un canal de comunicació que sovint està sotmès a soroll i que pot alterar el missatge transmès. El soroll d'un canal pot ésser de naturalesa diversa, però a efectes pràctics, el problema que suposa és que altera la informació transmesa. Detectar i corregir aquests errors és l'objecte dels codis correctors. La següent figura mostra de forma esquemàtica els diversos elements que intervenen en un procés de comunicació dotat d'un mecanisme de correcció d'errors:

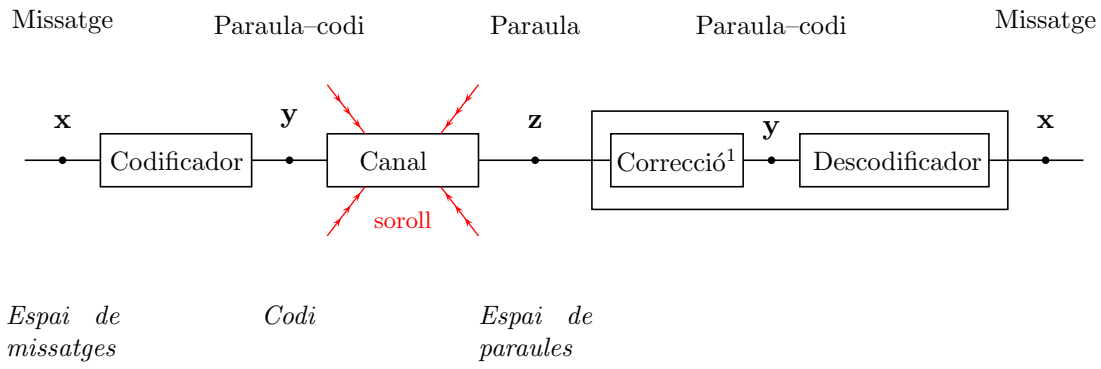

Figura 1.2: Esquema del procés de comunicació.

[1]Tot i que en la resta del text, s'entendrà descodificar una paraula com corregir-la, donat que, en la implementació s'anomena descodificar al fet de corregir una paraula, i que l'últim pas de descodificació acostuma a ser trivial.

La teoria de codis correctors cobreix aquests aspectes entre d'altres. Aquest treball es centra però en aquesta problemàtica; la transmissió de dades a través d'un canal de comunicació. Així doncs la teoria de la comunicació combina una elegant teoria matemàtica amb una important aplicació tecnològica.

La teoria de codis correctors fou originada per R. Hamming a finals dels anys '40. Hamming fou un matemàtic americà que va treballar per la companyia ≪Bell Telephone≫ fundada per A. G. Bell el 1877, on va col·laborar amb Claude E. Shannon ("pare" de la teoria de la informació). La motivació de Hamming era programar una computadora per corregir els errors que sorgien en els programes de les targetes perforades.

Existeixen multitud de codis correctors, cadascun amb les seves propietats, però ens centrarem únicament en els codis binaris donat que són codis que tenen una aplicació real en el món de les comunicacions digitals. Existeixen altres tipus de codis com els ternaris per exemple i que són útils a nivell teòric i matemàtic però que no tenen aplicació en l'àmbit de les comunicacions. Igualment, aquest treball parla dels codis com a *codis de bloc*, que són codis en que tots els seus elements —que posteriorment anomenarem paraules–codi— tenen la mateixa longitud.

L'aplicació dels codis correctors s'estén en diversos àmbits tecnològics i va guanyar importància i impuls degut a la necessitat de transmetre dades tals com imatges des de l'espai cap a la Terra. Però els codis correctors s'apliquen també sobre tecnologies més quotidianes com per exemple en els discs compactes.

Aquest treball descriu la implementació d'un tipus de codis correctors emprats a la realitat —es a dir no purament teòrics— anomenats *Codis de Golay* sobre un paquet de programari matemàtic de codi obert i lliure distribució anomenat SAGE. Un dels codis de Golay que serà presentat i estudiat en el capítol 4 va ser utilitzat per la NASA per transmetre imatges en color de Júpiter i Saturn des de les sondes espacials Voyager 1 i 2 el 1979 i 1980.

El paquet matemàtic SAGE per altra banda pretén ser una iniciativa lliure i oberta que aglutini les funcionalitats de part dels paquets d'anàlisi matemàtic, calculadors simbòlics i manipuladors algebraics; en molts casos propietaris, com Magma, Maple, Mathematica, Matlab,..., cobrint així diversos aspectes matemàtics incloent l'àlgebra, la combinatòria i el calcul. La iniciativa fou endegada per William Stein, professor de la Universitat de Washington, Seattle, USA. La primera versió va ser llançada el 24 de febrer de 2005 sota els termes de la GNU GPL (≪General Public License≫). A data d'avui va per la versió 4.3, compta amb al voltant de 150 persones que han contribuït directament amb codi i el projecte duu un ritme de llançaments de versió d'unes quatre setmanes per versió.

Capítol 2

Conceptes previs

Abans d'introduir els codis lineals, és necessari donar alguns conceptes previs aplicables als codis correctors en general. En aquest capítol es donaran algunes definicions necessàries per a la posterior definició i caracterització dels codis lineals, tals com la *distància de Hamming*, el *pes* d'una paraula o vector, o la *distància mínima* d'un codi. També s'introduiran aspectes associats als codis com el *radi de tangència* i el *radi de cobertura*.

Moltes de les definicions i resultats, d'aquest i els següents capítols, s'han extret de [BV01] i de [Hil93]. En quant a les demostracions que s'han transcrit, són en la seva majoria d'elaboració pròpia.

2.1 Distància de Hamming i Pes

Definició (Distància de Hamming)**.** Sigui A un alfabet i $\mathbf{x} = (x_1, \ldots, x_n)$ i $\mathbf{y} = (y_1, \ldots, y_n)$ dues paraules de A^n, es defineix la *Distància de Hamming*, com el nombre de coordenades en que $\mathbf{x}$ i $\mathbf{y}$ difereixen:

$$d(\mathbf{x}, \mathbf{y}) = |\{i : \ x_i \neq y_i\}|$$

Tot seguit s'enumeren les propietats de tota funció distància, que la de Hamming també compleix:

Siguin $\mathbf{x}, \mathbf{y}, \mathbf{z} \in A^n$,

(i) $d(\mathbf{x}, \mathbf{y}) \geq 0$ (definida positiva);

(ii) $d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$ (no–degenerada);

(iii) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ (simètrica);

(iv) $d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y})$ (propietat triangular).

Definició. Sigui $\mathbf{x} \in A^n$ i $C \subseteq A^n$, es defineix la distància $d(\mathbf{x}, C)$ com la mínima de les distàncies entre $\mathbf{x}$ i les paraules de C:

$$d(\mathbf{x}, C) = \min\{\, d(\mathbf{x},\mathbf{u}) :\ \mathbf{u} \in C \,\}$$

Definició (Distància mínima δ)**.** Sigui $C \subset A^n$ un codi, es defineix la *distància mínima* del codi com la mínima de les distàncies entre les possibles parelles de paraules del codi:

$$\delta(C) = \min\{\, d(\mathbf{u},\mathbf{v}) :\ \mathbf{u},\mathbf{v} \in C,\ \mathbf{u} \neq \mathbf{v} \,\}$$

Definició (Pes)**.** Sigui $\mathbf{0} \in A$, es defineix el *pes* $|\mathbf{x}|$ d'un vector $\mathbf{x}$ com el nombre de coordenades no nul·les de $\mathbf{x}$:

$$|\mathbf{x}| = |i :\ \mathbf{x}_i \neq 0|$$

Cal afegir que la distància entre dues paraules d'un codi (Distància de Hamming) és igual al pes de la diferència entre les dues paraules,

$$d(\mathbf{y}_1, \mathbf{y}_2) = |\mathbf{y}_1 - \mathbf{y}_2|$$

2.2 Radi de tangència i de cobertura

Sigui A un alfabet i $C \subset A^n$ un codi, es defineix el *radi de tangència* de C, denotat $\rho(C)$, com el màxim enter $r \geq 0$ tal que les boles de radi r centrades en les paraules de C siguin disjuntes dos a dos:

$$\rho(C) = \max\{\, r :\ B(\mathbf{u}, r) \cap B(\mathbf{v}, r) = 0, \text{ per a tot } \mathbf{u},\mathbf{v} \in C \,\}$$

El concepte dual és el *radi de cobertura* de C, denotat $\tau(C)$ i és defineix com el mínim enter $r \geq 0$ tal que les boles de radi r centrades en les paraules de C, recobreixen tot A^n:

$$\tau(C) = \min\{r :\ \bigcup_{\mathbf{u} \in C} B(\mathbf{u}, r) \supseteq A^n, \text{ per a tot } \mathbf{u} \in C\}$$

Donat que les boles de radi $\rho(C)$ centrades en les paraules de C, són disjuntes dos a dos, $\rho(C) \leq \tau(C)$. Quan $\rho(C) = \tau(C)$, el codi s'anomena *perfecte*.

Els codis de Golay, matèria sobre la qual es centra aquest treball, es caracteritzen per ser codis perfectes. En el capítol 4 s'amplia aquest concepte.

2.3 Capacitat detectora i capacitat correctora

En l'apartat anterior s'ha presentat el radi de tangència $\rho(C)$ d'un codi. Ara es veurà que aquest equival a la *capacitat correctora* del codi –és a dir a la capacitat que té un codi de corregir possibles errors deguts al canal en la transmissió d'un missatge– i que està relacionada amb la distància mínima. També es parlarà de la *capacitat detectora* –capacitat que té un codi de detectar errors deguts al canal.

Direm que un codi és t-corrector si qualsevol paraula amb t o menys errors pertany a una *única* bola amb centre en una paraula del codi amb radi $\leq \rho(C)$. Un cop rebuda una paraula $\mathbf{z}$ amb t o menys errors, l'algorisme de correcció retorna la paraula–codi del centre de la bola amb radi t que conté a $\mathbf{z}$. Així doncs el radi de tangència $\rho(C)$ és igual a la capacitat correctora t.

Direm que un codi C és s-detector quan en qualsevol bola de radi s amb centre en una paraula–codi, la única paraula–codi és precisament la del centre de la bola. Quan es rep una paraula $\mathbf{z}$ amb s o menys errors, l'algorisme de detecció consisteix en comprovar que $\mathbf{z} \in B(\mathbf{u}, s)$, per alguna $\mathbf{u} \in C$, i que $\mathbf{z} \neq \mathbf{u}$, pel que es determina que $\mathbf{z}$ no pertany al codi.

La relació entre aquestes capacitats de detecció i correcció i la distància mínima del codi, venen donades pels següents resultats.

Teorema 2.1. *(i) Un codi C és capaç de detectar fins a s errors en una paraula–codi si $\delta(C) \geq s + 1$.*

(ii) Un codi C és capaç de corregir fins a t errors en una paraula–codi si $\delta(C) \geq 2t+1$.

Corol·lari. *(i) C detecta fins a $\delta(C) - 1$ errors.*

(ii) C corregeix fins a $t = \rho(C) = \left\lfloor \dfrac{\delta - 1}{2} \right\rfloor$ errors.

És important assenyalar que les capacitats detectora i correctora d'un codi són magnituds inverses, en el sentit que si es maximitza la capacitat detectora, la capacitat correctora es veu minimitzada i a l'inrevés. Tal i com es veurà en la secció 3.5.3 del capítol 3, els codis amb distància mínima parell poden corregir t errors i detectar-ne $t + s$ (on $s = 1$) simultàniament, és a dir una combinació correcció i detecció maximitzant la correcció.

Capítol 3

Codis lineals

Vistos ja alguns dels conceptes previs necessaris, en aquest capítol s'introduiran els codis lineals com a subespais vectorials sobre cossos finits. Els codis lineals afegeixen estructura algebraica als codis i ens aporten els avantatges de l'àlgebra lineal. Aquests codis s'utilitzaran en els procediments de codificació i descodificació.

3.1 Definició i nomenclatura

Definició (Codi lineal)**.** Un $[n, k, \delta]_q$*–codi* C és un subespai vectorial del $\mathbb{F}_q$-espai vectorial $\mathbb{F}_q^n = \mathbb{F}_q \overset{n)}{\times} \ldots \times \mathbb{F}_q$, on $\mathbb{F}_q$ és un cos finit, pel que q té que ser potència d'un nombre primer ($q = p^m$, p és primer i $m \in \mathbb{Z}_{>0}$).

Els paràmetres que regeixen tot codi lineal són la longitud n, la dimensió k i distància mínima δ. La longitud del codi indica el nombre de coordenades dels vectors del codi, d'ara en endavant *paraules–codi*. La distància mínima δ, és com ja s'ha vist, la mínima de les distàncies entre les possibles parelles de paraules–codi.

Per últim, la *dimensió* k, és el nombre de vectors que conformen una base del codi, és a dir, el nombre de vectors linealment independents capaços de generar tot el codi.

Si es calculen totes les combinacions lineals dels vectors de la base, s'obté que la *mida* M del codi —el nombre de vectors que conté el codi— és exactament $M = q^k$. Efectivament, sigui $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ una base de C, es a dir, $\forall \mathbf{u} \in C : \ \mathbf{u} = \lambda_1 \mathbf{u}_1 + \ldots + \lambda_k \mathbf{u}_k$, amb $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_q$, aleshores hi ha tantes combinacions lineals com vectors $(\lambda_1, \ldots, \lambda_k)$. Això és el nombre de variacions amb repetició: $VR_{q,k} = q^k = |C|$. S'acostuma a escriure $C = \langle \mathbf{u}_1, \ldots, \mathbf{u}_k \rangle$.

Sigui C un $[n, k, \delta]_q$–codi, es defineixen els següents espais vectorials:

- $\mathbb{F}_q^k$: anomenat *espai de missatges*,
- $\mathbb{F}_q^n$: anomenat *espai de paraules*,
- $\mathbb{F}_q^{n-k}$: anomenat *espai de síndromes*, i com és habitual anomenar *redundància* al valor $r = n - k$, l'espai de síndromes també es pot denotar com $\mathbb{F}_q^r$.

Proposició 3.1. *El càlcul de la distància mínima en els codis lineals es redueix a buscar el menor dels pesos dels vectors no nuls del codi:*

$$\delta(C) = \min\{\, |\mathbf{u}| : \ \mathbf{u} \neq \mathbf{0},\ \mathbf{u} \in C \,\}$$

Demostració. Entre les possibles combinacions lineals dels vectors de la base sempre apareixerà el vector $\mathbf{0}$, $\lambda_1 = \ldots = \lambda_k = 0$, es a dir que sempre $\mathbf{0} \in C$. Així doncs la distància mínima es correspon a la mínima de les distàncies entre el vector $\mathbf{0}$ i la resta de vectors del codi:

$$\delta(C) = \min\{\, d(\mathbf{0}, \mathbf{x}),\ \mathbf{x} \in C \setminus \{\mathbf{0}\}\}$$

La qual cosa és equivalent a buscar el menor dels pesos dels vectors no nuls del codi. □

Aquesta relació, simplifica el càlcul de la distància mínima, reduint-lo a avaluar els pesos de les $M - 1$ *paraules–codi* no nul·les.

S'ha vist que un codi es pot definir donant-ne una base generadora del subespai vectorial $C \subsetneq \mathbb{F}_q^n$, tanmateix també es pot definir donant-ne una base del subespai ortogonal. En el primer dels casos s'obté una matriu anomenada *matriu generadora* del codi, en el segon el que s'obté és una *matriu de control* del codi.

3.2 Matriu generadora

Com ja s'ha anticipat, una *matriu generadora* del codi C, és una matriu $G \in M_{k\times n}(\mathbb{F}_q)$ les files de la qual formen una base de C. Cal adonar-se que aquesta matriu no és única donat que C admet més d'una base.

Aquesta matriu generadora, ens serà útil sobretot a efectes de codificació, tot i que també ho serà en descodificació. Escollir però una bona matriu no és trivial donat que certes matrius aporten avantatges tal i com es veurà tot seguit.

Una matriu G, pot ser transformada a una matriu equivalent G' a partir de *transformacions elementals*. Aquestes són:

(i) permutació de files de G;

(ii) multiplicació d'un escalar no nul per una fila;

(iii) sumar a una fila una combinació lineal de les altres.

Si la matriu generadora G és de la forma $\left(I_k \,\middle|\, A\right)$, on I_k és la matriu identitat, direm que aquesta és una *matriu sistemàtica* i aportarà avantatges tant en la codificació com en la descodificació.

3.3 Codificació

Donat un $[n,k]$–codi C amb matriu generadora G, es codifica un vector $\mathbf{x} \in \mathbb{F}_q^k$ pertanyent a l'espai de missatges, mitjançant l'aplicació:

$$\begin{array}{rccc} f_G: & \mathbb{F}_q^k & \longrightarrow & \mathbb{F}_q^n \\ & \mathbf{x} & \longmapsto & \mathbf{x}G = \mathbf{y} \in C \end{array}$$

El resultat és una *paraula–codi* $\mathbf{y} \in C \subseteq \mathbb{F}_q^n$. L'aplicació f_G és injectiva (la codificació és única) i no–exhaustiva ($\mathrm{im}(f_G) = C \subsetneq \mathbb{F}_q^n$).

Arribats a aquest punt, es pot observar que disposar d'una matriu G sistemàtica servirà per a que les k primeres coordenades de la paraula $\mathbf{y}$ siguin precisament el vector $\mathbf{x}$ (i.e. $\mathbf{y} = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_n)$), als que anomenarem *dígits de missatge*. Els $r = n-k$ dígits restants suposen la redundància afegida pel codi per tal de poder detectar i corregir errors i són anomenats *dígits de control*.

S'anomena *relació* o *ràtio* del codi a la relació entre la longitud del missatge a codificar i la de les paraules–codi: $R = \dfrac{n}{k}$ (on $k < n$).

3.4 Codi dual i matriu de control

Ja s'ha vist que un codi es pot definir donant-ne una matriu generadora. Tanmateix existeix una altra forma important per definir-lo, mitjançant l'anomenada *matriu de control* del codi. Si la matriu generadora ens és útil sobretot a efectes de codificació, la matriu de control ho és en descodificació.

Definició. Dos vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_\mathbf{q}^\mathbf{n}$ són ortogonals si el seu producte escalar és zero:

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n = \mathbf{0}$$

Definició (Codi dual)**.** Sigui C un $[n,k]_q$–codi, s'anomena *codi dual* de C denotat $C^\perp$, al format pel conjunt de vectors que són ortogonals a cadascuna de les paraules–codi de C. Així $C^\perp$ és un $[n, n-k]_q$–codi:

$$C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n : \ \mathbf{u} \cdot \mathbf{v} = \mathbf{0}, \ \forall \mathbf{u} \in C\}$$

Definició (Matriu de control)**.** Una matriu $H \in M_{r\times n}(\mathbb{F}_q)$ generadora del codi dual $C^\perp$, s'anomena *matriu de control* de C i satisfà la següent igualtat:

$$GH^t = \big(\mathbf{0}\big),$$

on $\big(\mathbf{0}\big)$ és la matriu nul·la de $M_{k\times r}(\mathbb{F}_q)$.

Com H es la matriu generadora del codi $C^\perp$, dual del C, podem donar una definició d'aquest a partir de H,

$$C = \{\mathbf{y} \in \mathbb{F}_q^n :\ \mathbf{y}H^t = \mathbf{0}\}.$$

El que equival a dir que les files de H són els coeficients d'un sistema homogeni de r equacions i n incògnites, les solucions del qual són els vectors de C.

Veiem a continuació que disposar d'una matriu generadora G en forma sistemàtica, servirà també per obtenir la matriu de control de forma immediata:

$$\text{Si } G = \big(I_k \big| A\big) \Longrightarrow\ H = \big(-A^t \big| I_r\big)$$

Notar que en codis binaris es pot obviar el signe negatiu (i.e. $-A = A$).

La matriu de control servirà també com una alternativa per al càlcul de la distància mínima del codi $\delta(C)$ ja que es pot calcular a partir dels vectors columna de la matriu de control. Així doncs la distància mínima es correspon al mínim nombre de columnes de H linealment dependents. Això evita haver de calcular els pesos de totes les paraules no–nul·les del codi.

A continuació es veu de forma esquemàtica la relació entre un codi i el seu codi dual:

$$\begin{array}{rcll} C = [n,k]_q & \longrightarrow & G,\, H & |C| = q^k \\ C^\perp = [n,r]_q & \longrightarrow & H,\, G & |C^\perp| = q^r \end{array}$$

3.5 Descodificació

En la transmissió d'una paraula–codi $\mathbf{y} \in C$ a través d'un canal, existeix la possibilitat que la paraula rebuda $\mathbf{z} \in \mathbb{F}_q^n$ contingui errors deguts al soroll del canal. Així doncs, el procés de descodificació consisteix en recuperar la paraula $\mathbf{y}$ tramesa. Aquest procés i en funció de la capacitat correctora t del codi permetrà corregir determinat nombre d'errors ($\leq t$) en la paraula $\mathbf{z}$ rebuda. Això serà possible gràcies a la redundància afegida al missatge en el procés de codificació.

Es defineix el vector d'error $\mathbf{e} = (e_1, \ldots, e_n)$ atribuït al canal com:

$$\mathbf{e} = \mathbf{z} - \mathbf{y}$$

El procés de descodificació es basa en un esquema de decisió per mínima distància, es a dir rebuda una paraula $\mathbf{z}$, aquesta s'interpreta (o corregeix) com la paraula $\mathbf{y} \in C$ a distància mínima de $\mathbf{z}$ en el cas que sigui única[1]. En cas de no ser-ho es dóna una situació de no–descodificació degut a que es produeix ambigüitat en el procés de decisió. En el millor dels casos, $d(\mathbf{z},\mathbf{y}) = 0$ i la descodificació és immediata ja que $\mathbf{z}=\mathbf{y} \in C$, es a dir, la paraula rebuda no conté errors. En el pitjor però, la paraula $\mathbf{z}$ contindria tants errors que s'escaparia de la bola amb radi $\rho(C)$ centrada en la paraula $\mathbf{y}$ corresponent; fins i tot podent convertir-se en una altra paraula del codi i donant lloc a una descodificació errònia. S'anomenen *errors residuals* a aquest tipus d'errors i són impossibles de detectar i corregir.

Abans d'entrar en detall en els mecanismes emprats pels algorismes de decisió, és necessari donar algunes definicions:

Definició (Classe lateral o coset)**.** Sigui un $[n, k]$–codi C sobre $\mathbb{F}_q^n$ i $\mathbf{a} \in \mathbb{F}_q^n$ un vector qualsevol, s'anomena *classe lateral o coset* $\mathbf{a} + C$, al conjunt:

$$\mathbf{a} + C = \{\mathbf{a} + \mathbf{y} : \ \mathbf{y} \in C\}$$

Teorema 3.2. *Sigui C un [n,k]-codi sobre $\mathbb{F}_q^n$,*

(i) qualsevol vector de $\mathbb{F}_q^n$ està en algun coset de C,

(ii) cada coset conté exactament q^k vectors,

(iii) dos cosets són o bé idèntics o bé totalment disjunts.

Així doncs la unió dels diferents cosets dóna lloc a l'espai $\mathbb{F}_q^n$ (el conjunt de cosets forma una *partició* de $\mathbb{F}_q^n$) i el nombre de cosets diferents és q^r.

Definició. Un vector de pes mínim d'un coset és anomenat *líder* del coset. En cas que hi hagi més d'un vector de pes mínim, se n'escull un aleatòriament.

Cal matisar sobre la definició anterior que en un codi t-corrector si $d(\mathbf{a}, C) \leq t$, cada coset $\mathbf{a}+C$ te un únic líder $\mathbf{e}$ de pes $|\mathbf{e}| \leq t$. Aquest fet és d'especial importància per la descodificació, ja que de no ser així i tal i com es veurà a continuació, la descodificació resultaria ambigua.

Corregir (o descodificar) una paraula $\mathbf{z}$ amb l'esquema de correcció per mínima distància, consisteix en trobar el líder $\mathbf{e}$ del coset $\mathbf{z}+C$ i corregir $\mathbf{z}$ com $\mathbf{y} = \mathbf{z} - \mathbf{e}$. El líder es corespon al vector d'error i el seu pes $|\mathbf{e}|$ és el nombre d'errors que contenen les paraules del seu coset i per tant el nombre d'errors que conté $\mathbf{z}$.

Per tal de construir algorismes per a la descodificació, s'empren algunes estructures i mecanismes com la *taula de Slepian* o la *síndrome* d'un vector.

[1] Veurem que en els codis perfectes la paraula $\mathbf{y}$ sempre és única ja que $\rho(C) = \tau(C)$. El codi binari de Golay n'és un exemple.

3.5.1 Taula de Slepian

Una taula de Slepian per a un $[n, k]$–codi C és una $(q^k \times q^r)$-taula que conté tots els cosets d'un codi llistats per files, la primera columna de la qual conté els líders dels cosets i la primera fila els vectors del codi. L'objectiu de la taula és trobar l'error $\mathbf{e}$ introduït pel canal. Aquest error es correspon al líder del coset on pertany la paraula $\mathbf{z}$ rebuda. Tot seguit es descriu el procediment per a la construcció de la taula de Slepian.

Pas 1. Llistar en la primera fila les paraules del codi, començant pel vector $\mathbf{0}$.

Pas 2. Prendre un vector $\mathbf{a} \in \mathbb{F}_q^n$ de pes mínim que no aparegui a la taula i llistar-ne el coset $\mathbf{a} + C$ en la segona fila, col·locant $\mathbf{a} + \mathbf{y}_i$ sota de cada $\mathbf{y}_i$, per a cada $\mathbf{y} \in C$.

Pas 3. Escollir un nou vector $\mathbf{a}'$ que no aparegui anteriorment i de pes mínim. Llistar-ne novament el coset $\mathbf{a}' + C$ en la següent fila.

Pas 4. Seguir fins haver llistat tots els cosets i cada vector de $\mathbb{F}_q^n$ aparegui exactament un cop.

L'algorisme de descodificació recorre la taula cercant la paraula $\mathbf{z}$ rebuda. Un cop és trobada, es pren com a vector d'error $\mathbf{e}$ el *líder* del coset al que pertany $\mathbf{z}$ —corresponent al primer vector de la fila— i es descodifica com

$$\mathbf{y} = \mathbf{z} - \mathbf{e}.$$

Exemple 3.3. Suposem un $[4, 2]_2$–codi lineal C amb matriu generadora $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$. Comencem llistant en la primera fila de la taula totes les paraules del codi C, es a dir les combinacions lineals dels vectors de la matriu G, començant pel $\mathbf{0}$ i seguim com s'indica en el procediment anterior:

0 0 0 0	1 0 1 1	0 1 0 1	1 1 1 0
1 0 0 0	0 0 1 1	1 1 0 1	0 1 1 0
0 1 0 0	1 1 1 1	0 0 0 1	1 0 1 0
0 0 1 0	1 0 0 1	0 1 1 1	1 1 0 0

Taula 3.1: Taula de Slepian

Notar que cada entrada de la taula de Slepian és la suma de la paraula–codi del cap de la corresponent columna amb el líder a l'extrem esquerra de la fila.

Suposem que es rep la paraula $\mathbf{z} = (1, 1, 1, 1)$, aleshores l'algorisme recorre la taula fins trobar la paraula rebuda i en determina l'error corresponent al líder $\mathbf{e} = (0, 1, 0, 0)$, descodificant $\mathbf{z}$ com la paraula $\mathbf{y} = \mathbf{z} - \mathbf{e} = (1, 0, 1, 1)$ corresponent al cap de la columna que conté $\mathbf{z}$.

Aquest mecanisme de descodificació, però, té importants deficiències si s'aplica sobre codis grans, degut als requeriments d'emmagatzematge necessaris i al temps requerit en recórrer la taula. Una millora d'aquest procediment consisteix en llistar només el coset $\mathbf{z} + C$, i obtenir així directament la fila on es troba $\mathbf{z}$ (ja que $\mathbf{0} \in C$ i $\mathbf{z}+\mathbf{0} = \mathbf{z}$), prendre el vector de la fila de pes mínim com a líder $\mathbf{e}$ i retornar $\mathbf{y} = \mathbf{z}-\mathbf{e}$.

Un sistema més sofisticat de descodificació es descriu a continuació, és l'anomenada descodificació *via síndrome*.

3.5.2 Descodificació via síndrome

Definició (síndrome)**.** Sigui C un $[n, k]$–codi lineal definit per la matriu de control H, es defineix la *síndrome* d'una paraula $\mathbf{z} \in \mathbb{F}_q^n$ com l'aplicació

$$\begin{array}{rccl} f_{H^t}: & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^r \\ & \mathbf{z} & \longmapsto & \mathbf{z}H^t = s(\mathbf{z}) \in \mathbb{F}_q^r \end{array}$$

L'aplicació f_{H^t} és no–injectiva (diferents paraules tenen la mateixa síndrome) i exhaustiva (es cobreix tot l'*espai de síndromes*).

- Les paraules de C, són les que tenen síndrome $\mathbf{0}$: $\mathbf{z} \in C \Leftrightarrow s(\mathbf{z}) = \mathbf{0}$.
- Dues paraules pertanyen al mateix coset de C si, i només si, tenen la mateixa síndrome. Conseqüentment, totes les paraules d'un coset tenen la mateixa síndrome.

La síndrome d'una paraula dóna informació sobre l'error que la paraula conté i és utilitzada per tant per diversos mecanismes de correcció. La síndrome depèn únicament de l'error i no de la paraula–codi i això implica que cada coset tingui la mateixa síndrome associada. Aquest fet es demostra a través les propietats de L'Àlgebra lineal, sobre les quals es basen els codis lineals:

Sigui $\mathbf{y} \in C$ *una paraula–codi,* $\boldsymbol{e}$ *un vector d'error i* $\boldsymbol{z}$ *una paraula rebuda,*

$$\begin{array}{ccccccc} (\mathbf{y}+\mathbf{e}) = \mathbf{z} & \Longrightarrow & (\mathbf{y}+\mathbf{e})H^t = \mathbf{z}H^t & \Longrightarrow & \mathbf{y}H^t + \mathbf{e}H^t = \mathbf{z}H^t & \Longrightarrow \\ & \Longrightarrow & s(\mathbf{y}) + s(\mathbf{e}) = s(\mathbf{z}) & \Longrightarrow & s(\mathbf{e}) = s(\mathbf{z}) & & \end{array}$$

I no només això, sinó que també aporta informació sobre on s'han produït els errors tal com es mostra al següent teorema (en [CHP03], obtingut de `Google-books`).

Teorema 3.4. *Sigui* C *un* $[n, k]_q$*–codi lineal* t*-corrector, amb matriu de control* $H = \left(-A^t | \, I_r\right)$*, i sigui* $\mathbf{z} = \mathbf{y} + \mathbf{e}$ *una paraula amb* $\mathbf{y} \in C$ *i* $|\mathbf{e}| \leq t$*, aleshores, els dígits de missatge de la paraula* $\mathbf{y}$ *són correctes, si i només si,* $|s(\mathbf{y})| \leq t$*.*

Així doncs pel teorema 3.4, quan el pes de la síndrome d'una paraula $\mathbf{z}$ rebuda és inferior a la capacitat correctora (i major que zero), els errors que conté $\mathbf{z}$, estan en les posicions redundants (dígits de control).

La descodificació via síndrome, s'aplica sobre la taula de Slepian i l'eficiència del mètode rau en el fet que per trobar el vector d'error, no és necessari recórrer tota la taula de Slepian. Calculant la síndrome de la paraula a descodificar es coneix a quin coset pertany (a quina fila de la taula) i per tant se'n coneix també el líder $\mathbf{e}$. Així doncs aquest mecanisme simplifica la cerca del líder del coset on pertany la paraula a descodificar, produint un estalvi en el temps requerit en recórrer la taula i en la memòria requerida al ser únicament necessari emmagatzemar els líders dels cosets i les respectives síndromes. A aquesta taula se l'anomena *taula de cerca de síndromes*.

3.5.3 Descodificació incompleta

Els dos mecanismes de descodificació anteriors funcionen correctament sempre i quan el nombre d'errors produïts en la paraula $\mathbf{z}$ rebuda sigui inferior o igual a la capacitat correctora t del codi, donat que en aquestes circumstàncies, $\mathbf{z}$ està dins de la bola amb radi t i centre en una paraula $\mathbf{y} \in C$ (i.e. $\mathbf{z}$ està a distància mínima únicament de $\mathbf{y}$).

Però en el cas de que un nombre major d'errors es produeixi, $\mathbf{z}$ cau fora del radi de tangència de $\mathbf{y}$ i es pot produir ambigüitat a l'hora de descodificar ja que $\mathbf{z}$ pot estar a igual distància mínima d'una altra paraula $\mathbf{y}' \in C$. En aquest cas és necessari demanar la retransmissió de la paraula.

La *descodificació incompleta* és una mescla de correcció i detecció d'errors. Es tracta d'un esquema a través del qual es garanteix la correcció de fins a t errors en qualsevol paraula–codi i també la detecció en alguns casos de més de t errors. Per tal d'aconseguir-ho, es genera la taula de Slepian, llistant només les files corresponents als cosets amb líder $\mathbf{e}$ de pes $|\mathbf{e}| \leq t$, donat que les paraules de la resta de cosets contindran un nombre d'errors superior a la capacitat correctora t del codi. En cas de rebre una paraula que no aparegui a la taula, es demana la retransmissió de la paraula.

Si s'empra la descodificació incompleta mitjançant la taula de cerca de síndromes, es pot prescindir de la taula de Slepian completament (no és necessari llistar els cosets per coneixe'n els líders). Això es deu a que es coneixen exactament quins són els líders dels cosets de pes $\leq t$. Així doncs es poden escriure directament tots els líders de pes $\leq t$ i calcular-ne les respectives síndromes per tal de construir la taula de cerca de síndromes, que serà finalment utilitzada en descodificació. Il·lustrem-ho amb un exemple:

Exemple 3.5. Sigui C un $[5, 2, 3]_2$–codi, amb matriu generadora $G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$ i matriu de control $H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$; $\delta(C) = 3$ per tant és un codi 1-corrector.

Construïm la taula de cerca de síndromes llistant els líders dels cosets de pes ≤ 1 en la primera columna i calculant-ne els corresponents síndromes en la segona columna:

líder **e** del coset	síndrome s(**z**)
0 0 0 0 0	0 0 0
1 0 0 0 0	1 1 0
0 1 0 0 0	0 1 1
0 0 1 0 0	1 0 0
0 0 0 1 0	0 1 0
0 0 0 0 1	0 0 1

Taula 3.2: Taula de cerca de síndromes

En rebre una paraula $\mathbf{z}$, se'n calcula la síndrome. En cas que no aparegui a la taula es demana retransmissió de la paraula (ja que conté més de t errors). Si la síndrome apareix a la taula es descodifica com $\mathbf{y} = \mathbf{z} - \mathbf{e}$.

Tal com es veu no ha estat necessari llistar totes les paraules de cada coset, el qual hauria significat llistar 2^r cosets $\times$ 2^k paraules $= 32$ paraules.

Quan un codi té distància mínima parell $\delta(C) = 2t+2$, la descodificació incomplerta garanteix la correcció de fins a t errors i la detecció de $t+1$ errors simultàniament. Això és deu a que les paraules que cauen a distància $t+1$ d'una paraula–codi, no estan mai dins del radi de cobertura t de qualsevol altra paraula–codi, per tant són paraules que no descodifiquen; se'n demana retransmissió. La figura 3.1 il·lustra aquest fet.

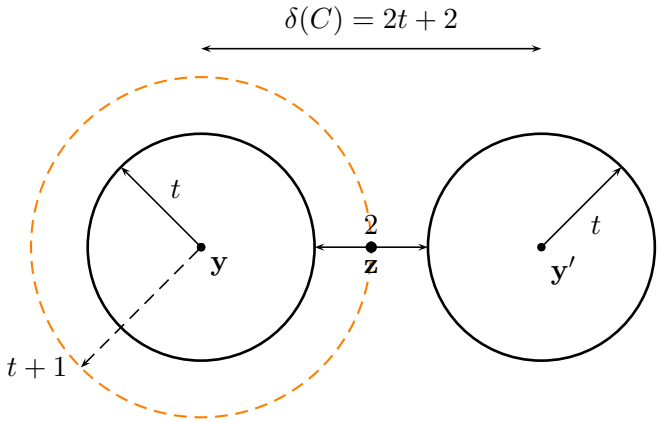

Figura 3.1: Representació esquema t-corrector i $t + s$ detector.

Capítol 4

Codis Perfectes i codis de Golay

En aquest capítol, es presentaran i descriuran els codis perfectes i els dos codis binaris de Golay (G_{24} i G_{23}) —un dels quals (G_{23}) és perfecte—, i que seran implementats en Sage en el capítol 6.

Es veurà que els codis perfectes sempre són capaços de descodificar "completament", ja que qualsevol paraula de l'espai de paraules està a distància mínima d'una única paraula–codi.

En aquest capítol es veurà també un dels algorismes de descodificació específic per als codis binaris de Golay i que millora substancialment els mecanismes de descodificació per a codis lineals presentats en el capítol anterior. El procediment i la millora es deuen a les propietats específiques d'aquests codis i de les matrius que els defineixen.

Es deixen de banda però, els dos codis ternaris de Golay ja que s'escapen de l'àmbit d'aquest treball i no tenen aplicació en la transmissió de dades a través de canals de comunicació digitals.

4.1 Codis perfectes

Definició. Sigui C un $[n,k]_q$–codi, es defineix el *volum* d'una bola amb radi t i centre en una paraula–codi com,

$$V_t = \sum_{i=0}^{t} \binom{n}{i} (q-1)^i,$$

i equival al nombre de paraules–codi tancades en la bola de radi t. (Notar que per als codis binaris $(q-1)^i = 1,\ \forall i$).

Teorema 4.1 (fita de Hamming)**.** *Sigui M la mida d'un codi C de longitud n i V_t el volum de les boles de radi t, el producte $M \cdot V_t$ mai pot ser estrictament superior a $\mathbb{F}_q^n$, per tant,*

$$M \leq \frac{q^n}{V_t}.$$

S'anomena a aquesta desigualtat fita de Hamming.

Cal notar que tant la definició com el teorema anteriors són vàlids per a codis més generals, no estrictament lineals, sobre alfabets q–aris.

Per tal que un codi pugui corregir t errors, les boles de radi t centrades en les paraules–codi han de ser disjuntes dos a dos —com ja és sabut— i per a que sigui perfecte, cal que cobreixin tot l'espai $\mathbb{F}_q^n$. Es a dir, *un codi és perfecte quan assoleix la fita de Hamming.* Una forma equivalent d'expressar-ho, és dir que un codi és perfecte quan el radi de tangència i el radi de cobertura coincideixen ($\rho(C) = \tau(C)$) o que les boles de radi t centrades en les paraules–codi formen una *partició* de $\mathbb{F}_q^n$.

Que un codi lineal assoleixi la fita de Hamming no és un fet trivial i existeixen pocs conjunts de paràmetres que donen lloc a un codi perfecte. La següent desigualtat s'assoleix també únicament quan un codi és perfecte i permet trobar els paràmetres de dit codi,

$$V_t \leq q^r \iff \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^r.$$

- Per a $t = 1$, tenim que:

  $$1 + n(q-1) = q^r \Longrightarrow n = \frac{q^r - 1}{q-1},$$

  es tracta dels anomenats *codis de Hamming*, amb paràmetres

  $$\mathrm{Ham}(r, q) = \left[\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1} - r, 3\right]_q \text{–codi},$$

  es a dir, codis lineals amb distància mínima $\delta = 3$ que existeixen sobre qualsevol cos finit $\mathbb{F}_q$.

- Per a $t = 2$, l'equació esdevé:

  $$1 + n(q-1) + \frac{n(n-1)}{2}(q-1)^2 = q^r \ ,$$

  i la única solució s'obté per $q = 3$, $n = 11$ i $r = 5$, el qual dóna lloc a l'anomenat codi ternari de Golay amb paràmetres $[11, 6]_3$ i que té distància mínima $\delta = 5$.

- Per a $t = 3$, l'equació esdevé:

$$1 + n(q-1) + \frac{n(n-1)}{2}(q-1)^2 + \frac{n(n-1)(n-2)}{6}(q-1)^3 = q^r \ ,$$

  i la solució, s'obté per $q = 2$, $n = 23$ i $r = 11$, donant lloc a l'anomenat codi binari de Golay amb paràmetres $[23, 12]_2$ i que té distància mínima $\delta = 7$.

I, com va demostrar A. Tietäváinen el 1973, aquests són tots els codis lineals perfectes que existeixen.

El fet que un codi sigui perfecte implica que no conté redundància inútil i el concepte es correspon a un criteri d'optimalitat[1].

Teorema 4.2. *La distància mínima d'un codi perfecte és senar.*

Demostració. Sigui C un $[n, k, \delta]_q$–codi perfecte, el radi de tangència de C és $\rho = \left\lfloor \frac{\delta - 1}{2} \right\rfloor$, d'on es dedueix que $\delta = 2\rho + 1$ o bé $\delta = 2\rho + 2$. Per a una demostració senzilla i visual referir-se a la figura 3.1 on es veu que si $\delta = 2\rho + 2$, les boles de radi ρ centrades en les paraules–codi no recobreixen tot l'espai $\mathbb{F}_q^n$ i queden buits. Per tant per a un codi perfecte, donat que $\rho = \tau$, δ només pot valer $2t + 1$. □

El fet que un codi sigui perfecte, aporta l'avantatge que cada coset té un únic líder (el pes del qual és $\leq t$) i per tant sempre descodifica.

4.2 Codis de Golay binaris

Com ja s'ha anat avançant, els codis de Golay són una classe de codis correctors perfectes. En particular la forma binària del codi de Golay és un dels tipus més importants de codis binaris lineals, ja que és un dels pocs exemples de codi no–trivial[2] perfecte. Es considera al G_{23} el *codi de Golay binari* i al G_{24} —l'extensió de l'anterior— el *codi de Golay binari estès*.

El codi de Golay $G_{23} = [23, 12, 7]_2$ és l'únic codi multicorrector binari perfecte i és capaç de corregir qualsevol combinació de tres o menys errors aleatoris en un bloc de 23 dígits. Aquest codi té una abundant estructura algebraica i des que fou descobert per M. J. Golay al 1949, ha estat objecte d'estudi per matemàtics i teòrics. El G_{23} ha estat emprat en diversos sistemes reals de comunicació. També cal assenyalar que el codi G_{24} fou emprat per la NASA en la transmissió de fotografies a color de Júpiter i Saturn entre les sondes Voyager i la Terra el 1979 i 1980. La transmissió

[1] Un codi optimal és aquell que té la mida màxima d'entre tots els possibles codis q-aris, fixades la q, la longitud n, i la distància mínima δ.

[2] Un codi trivial és aquell que només conté un element o bé aquell en que $n = k$.

d'imatges en color requeria tres vegades més d'informació a enviar que les imatges en escala de grisos. Per això es va utilitzar el G_{24}, que tot i ser 3-corrector, té una relació de transmissió més elevada que la d'altres codis utilitzats anteriorment pel mateix propòsit com el codi de Hadamard $[32, 6, 16]$ [3], utilitzat en la sonda Mariner 9 el 1971, que tot i tenir una capacitat correctora força superior (7-corrector), té una relació de transmissió molt baixa.

El G_{24} és una extensió del codi perfecte G_{23} que s'obté afegint un bit de paritat al G_{23}. Sigui G_{23} el codi de Golay $[23, 12, 7]_2$ i $\mathbf{y} = (y_1, y_2, \ldots, y_{23}) \in G_{23}$, la següent funció afegeix un bit de paritat a les paraules del codi:

$$\forall \mathbf{y} \in G_{23} \longrightarrow \hat{\mathbf{y}} = \begin{cases} (y_1, y_2, \ldots, y_{23}, 0) & \text{si } |\mathbf{y}| \text{ és parell} \\ (y_1, y_2, \ldots, y_{23}, 1) & \text{si } |\mathbf{y}| \text{ és senar} \end{cases} : \hat{\mathbf{y}} \in G_{24}.$$

Així, el G_{24} és un codi lineal amb paràmetres $[24, 12, 8]_2$. Tot i que la distància mínima del G_{24} és superior a la del G_{23}, tots dos són codis 3-correctors[4], però el G_{24} és també 4-detector.

G_{24} és un codi *quasi–perfecte*. Això significa que el radi de cobertura excedeix el radi de tangència en exactament una unitat: $\tau(G_{24}) = \rho(G_{24}) + 1$, es a dir qualsevol paraula de l'espai de paraules està com a màxim a distància $t + 1$ d'alguna paraula–codi (que no serà única al no ser perfecte). Un codi quasi–perfecte té la major part dels líders dels cosets de pes $\leq t$, alguns líders de pes $t + 1$ i cap de pes $> t + 1$. Concretament per al G_{24}, es tenen $\binom{24}{0} + \binom{24}{1} + \binom{24}{2} + \binom{24}{3} = 2325$ líders de pes ≤ 3 i $2^{12} - 2325 = 1771$ líders de pes 4.

Observant els paràmetres dels codis de Golay, en podem veure la *relació o "ràtio" de transmissió* i *l'índex de correcció*, o fracció de bit capaç de corregir, $\frac{\rho}{n}$,

- per al $G_{24} = [24, 12]_2$–codi es té,
  - relació de transmissió: $\frac{k}{n} = \frac{24}{12} = 0.5$,
  - índex de correcció: $\frac{3}{24} = 0.125$ (corregeix fins a un 12.5% d'errors);
- per al $G_{23} = [23, 12]_2$–codi es té,
  - relació de transmissió: $\frac{k}{n} = \frac{23}{12} = 0.52$,
  - índex de correcció: $\frac{3}{23} = 0.13$ (corregeix fins a un 13% d'errors).

[3] Els codis de Hadamard són $[2^n, n + 1, 2^{n-1}]$-codi.

[4] Existeixen però alguns algorismes que permeten corregir determinades combinacions o ràfegues de fins a 4 errors en el G_{24}.

Veiem que un missatge codificat ocupa el doble que el missatge original però per contrapartida, es guanya en capacitat correctora. Aquests índex de correcció són més que suficients tenint en compte les probabilitats d'error dels canals habituals de comunicació (e.g. probabilitats error: fibra òptica $\approx 10^{-9}$, ràdio $\approx 10^{-3}$).

Degut a les propietats algebraiques del G_{24} resulta més senzill descriure i construir primer aquest i perforar-lo suprimint-ne el bit de paritat per obtenir el G_{23}. Algunes de les propietats algebraiques del G_{24} es veuen tot seguit.

Definició (Matriu antiortogonal)**.** Sigui $A \in M_{k\times k}(\mathbb{F}_q)$,

$$A \text{ és } \textit{antiortogonal} \iff AA^t = -I_k.$$

Definició (Codi autodual)**.** Un codi C és *autodual* si i només si $C = C^\perp$; o de forma equivalent, si tota matriu generadora G, n'és també matriu de control.

Si C és autodual, la longitud del codi és el doble que la dimensió, ja que igualant les dimensions de C i $C^\perp$ tenim que, $k = n-k$, aleshores, $n = 2k$ (e.g. $G_{24} = [24, 12]_2$–codi).

Del fet que un codi sigui autodual, és a dir que tota matriu generadora és també matriu de control, i tal com s'ha vist en la definició de matriu de control en el capítol 3, tenim que,

$$GG^t = HH^t = GH^t = \begin{pmatrix} 0 \end{pmatrix}$$

Proposició 4.3. *Una matriu A és* antiortogonal *si, i només si, el codi que té per matriu generadora $G = (I_k|A)$ és un codi autodual.*

Proposició 4.4. *Sigui $A \in M_{k\times k}(\mathbb{F}_q)$ una matriu antiortogonal i simètrica. Considerem el $[2k, k]_q$–codi lineal C amb matriu generadora $G = (I_k|A)$ i $H = (-A^t|I_k)$. Aleshores, per a tot $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^k$,*

$$(\mathbf{x}||\mathbf{y}) \in C \iff (-\mathbf{y}||\mathbf{x}) \in C.$$

Demostració. Per la proposició 4.3 tenim que el codi C és un codi autodual, per tant les matrius G i H són ambdues generadores (i de control) de C. Donats $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^k$, el vector $(\mathbf{x}||\mathbf{y})$ pertany a C si, i només si, és combinació lineal de les files de G. La mateixa combinació lineal però amb les files de H dóna el vector $(-\mathbf{y}||\mathbf{x})$ que per tant també pertany a C. □

4.2.1 Construcció dels codis de Golay Binaris

Tal i com s'ha comentat, es construirà primer el codi binari de Golay estès G_{24} i posteriorment se'n realitzarà la perforació per obtenir el G_{23} tot eliminant-ne l'últim bit. Per tal de construir el G_{24}, a l'igual que qualsevol codi, n'hi ha prou amb donar-ne una matriu generadora. Tot i això, des del punt de vista del lector podria resultar

poc satisfactori donat que no queda clar d'on prové la matriu però prova almenys que el codi existeix. Una forma més natural de definir-los és com a codis cíclics, però aquesta definició de codis s'escapa de l'àmbit d'aquest treball. Així doncs es donarà la matriu generadora en la forma sistemàtica.

Considerem la següent matriu $A \in M_{12}(\mathbb{F}_2)$, simètrica i antiortogonal,

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

A més a més aquesta matriu posseeix altres propietats. La matriu A té 7 uns en totes les files a excepció de la última que en té 11; així doncs totes les files tenen un nombre imparell d'uns. També es pot observar que la submatriu A' obtinguda de suprimir la última fila i última columna de A és una matriu *circulant*, es a dir, cada fila és obtinguda a partir de l'anterior desplaçant-ne les seves coordenades una posició a l'esquerra (incloent la primera fila que s'obté de la última).

Així doncs es defineixen els dos codis de Golay binaris de la següent forma,

- El *codi de Golay binari estès* G_{24} es defineix mitjançant de la matriu generadora en forma sistemàtica
$$G = \Big(I_{12} \,\Big|\, A \Big)$$
- El *codi de Golay binari* G_{23} es defineix mitjançant la matriu generadora G' en forma sistemàtica que s'obté al perforar G_{24} per l'última columna, es a dir, eliminant de la matriu G la última columna.

De forma equivalent, el codi G_{24} es pot obtenir afegint un bit de paritat a les files de la matriu generadora del G_{23}; s'anomena a aquest fet *extendre el codi*.

Degut a la simetria de la matriu A i al cos sobre el que viuen els codis binaris, es pot fer una petita simplificació en l'expressió de la matriu de control de G_{24},

$$A^t = A \quad \text{i} \quad -x = x \in \mathbb{F}_2, \text{ per tant, } H = \big(A \,\big|\, I_{12}\big).$$

Cal recalcar que el G_{24} al ser autodual, les dues matrius $G = (I_{12} \,|\, A)$ i $H = (\, A \,|\, I_{12})$ són simultàniament generadores i de control.

Teorema 4.5. *Sigui C un codi lineal,*
(i) Si C és auto-ortogonal[5] *i té una matriu generadora on cadascuna de les files té pes divisible per 4, aleshores totes les paraules–codi de C tenen pes divisible per 4.*
(ii) Si cada paraula–codi de C té pes divisible per 4, aleshores C és auto-ortogonal.

El codi G_{24}, verifica el teorema 4.5, a més a més, si $\mathbf{y}, \mathbf{y}' \in G_{24}$, llavors $\mathbf{y}$ i $\mathbf{y}'$ difereixen en almenys les últimes vuit coordenades. Equivalentment, qualsevol vector de G_{24} a excepció del $\mathbf{0}$, té al menys vuit coordenades que no són zero.

Degut a que donat un vector $\mathbf{v} \in \mathbb{F}_2^{24}$ ens interessarà distingir la meitat esquerra (les 12 primeres coordenades) de la meitat dreta (les 12 darreres coordenades), es denota $\mathbf{v}_L = (v_1, \ldots, v_{12})$ a la meitat esquerra i $\mathbf{v}_R = (v_{13}, \ldots, v_{24})$ a la meitat dreta.

Lema 4.6. *El codi G_{24} no té cap paraula–codi de pes 4.*

Demostració. Sigui $\mathbf{y} \in G_{24}$ i suposeu que $\mathbf{y}$ té pes 4. Aleshores alguna de les següents situacions s'hauria de donar:

- $|\mathbf{y}_L| = 0$, $|\mathbf{y}_R| = 4$. Això no és possible donat que es dedueix a partir de la matriu generadora G que $\mathbf{0}$ és l'única paraula–codi amb $|\mathbf{y}_L| = 0$.
- $|\mathbf{y}_L| = 1$, $|\mathbf{y}_R| = 3$. Si $|\mathbf{y}_L| = 1$, aleshores $\mathbf{y}$ és una de les files de G, cap de les quals té $|\mathbf{y}_R| = 3$.
- $|\mathbf{y}_L| = 2$, $|\mathbf{y}_R| = 2$. Si $|\mathbf{y}_L| = 2$, aleshores $\mathbf{y}$ és la suma de dues files de G, però es pot observar que cap suma de dos files de la matriu A té pes 2.
- $|\mathbf{y}_L| = 3$, $|\mathbf{y}_R| = 1$. Aquesta situació és equivalent a la segona prenent la matriu $H = (A|\, I_{12})$ com a generadora. La paraula–codi $\mathbf{y}$ hauria de ser una de les files de H, cap de les quals té $|\mathbf{y}_L| = 3$
- $|\mathbf{y}_L| = 4$, $|\mathbf{y}_R| = 0$. Aquesta situació és equivalent a la primera. Prenent un altre cop H com a a generadora, es dedueix que $\mathbf{0}$ és la única paraula–codi amb $|\mathbf{y}_R| = 0$.

□

Lema 4.7. *La distància mínima de G_{24} és 8.*

Demostració. Pel teorema 4.5 les paraules–codi de G_{24} han de tenir pes 4,8,12,... i pel lema 4.6 no hi ha cap paraula–codi amb pes 4. Observant la matriu generadora $G = (I_{12}|A)$, es veu que la primera fila té pes 8, per tant $\delta(G_{24}) = 8$. □

Lema 4.8. *La distància mínima de G_{23} és 7.*

[5] Un codi és auto-ortogonal quan $C \subseteq C^{\perp}$. Els codis autoduals són per tant auto-ortogonals.

Demostració. S'ha vist quan s'han presentat els codis perfectes en la secció 4.1 que el codi binari de Golay G_{23} era un $[23, 12]$–codi amb capacitat correctora $t = 3$. Per tant la distància mínima és, $\delta = 2t + 1 = 7$ o bé, $\delta = 2t + 2 = 8$. Pel teorema 4.2, la distància mínima de G_{23} només pot ser 7. Això es pot demostrar igualment observant la matriu generadora de G_{23}, on es veu que la primera fila té pes 7. Per tant $\delta(G_{23}) = 7$. □

La taula 4.1 mostra el pes de les paraules–codi de G_{23} i de G_{24}.

Pes	**Nombre de paraules–codi**	
	$\mathbf{G_{23}}$	$\mathbf{G_{24}}$
0	1	1
7	253	0
8	506	759
11	1288	0
12	1288	2576
15	506	0
16	253	759
23	1	0
24	0	1
Total	4096	4096

Taula 4.1: Pesos de les paraules–codi de G_{23} i G_{24}

4.2.2 Descodificació dels codis de Golay binaris

El G_{24} conté $2^{12} = 4096$ paraules–codi i l'espai de paraules $\mathbb{F}_2^{24}$ descompon en $2^{24}/2^{12} = 2^{12} = 4096$ cosets. Així doncs l'ús de la taula de Slepian o inclús de la taula de cerca de síndromes resultaria molt costós i per tant el mecanisme genèric de descodificació presentat en el capítol 3 és poc eficient. Degut a les propietats algebraiques d'aquests codis, existeix un mecanisme de descodificació específic que agilitza enormement el procés.

Com s'ha vist, tant G com H poden ser matrius de control. En aquesta situació, es pot definir per tant la síndrome d'una paraula de dues maneres. Sigui $\mathbf{z} \in \mathbb{F}_2^{24}$,

$$S_H(\mathbf{z}) = \mathbf{z}H^t = (\mathbf{z}_L||\mathbf{z}_R)\begin{pmatrix} A \\ I_{12} \end{pmatrix} = \mathbf{z}_L A + \mathbf{z}_R$$

$$S_G(\mathbf{z}) = \mathbf{z}G^t = (\mathbf{z}_L||\mathbf{z}_R)\begin{pmatrix} I_{12} \\ A \end{pmatrix} = \mathbf{z}_L + \mathbf{z}_R A$$

i atès que $A^2 = AA^t = -I_{12} = I_{12}$, es dóna la següent relació entre S_H i S_G:

$$S_H(\mathbf{z})A = (\mathbf{z}_L A + \mathbf{z}_R)A = \mathbf{z}_L A^2 + \mathbf{z}_R A = \mathbf{z}_L + \mathbf{z}_R A = S_G(\mathbf{z})$$

Proposició 4.9. *Siguin $\mathbf{y} \in G_{24}$ una paraula–codi, $\mathbf{e} \in \mathbb{F}_2^{24}$ un vector d'error amb pes $|\mathbf{e}| \leq 3$ i $\mathbf{z} = \mathbf{y} + \mathbf{e} \in \mathbb{F}_2^{24}$ una paraula rebuda; i siguin $\mathbf{u}_i$ l'i-èsim vector fila de la matriu A i $\mathbf{e}_i$ l'i-èsim vector fila de I_{12}, es dóna almenys una de les situacions següents,*

$$\begin{array}{lllll}
(i) & S_H(\mathbf{y}) \leq 3 & \Longrightarrow & \mathbf{y} = \mathbf{z} + (\mathbf{0} \,||\, S_H(\mathbf{y})) \\
(ii) & 0 \leq |S_H(\mathbf{y}) + \mathbf{u}_i| \leq 2 & \Longrightarrow & \mathbf{y} = \mathbf{z} + (\mathbf{e}_i \,||\, S_H(\mathbf{z}) + \mathbf{u}_i) \\
(iii) & |S_G(\mathbf{y})| \leq 3 & \Longrightarrow & \mathbf{y} = \mathbf{z} + (S_G(\mathbf{y}) \,||\, \mathbf{0}\,) \\
(iv) & 0 \leq |S_G(\mathbf{y}) + \mathbf{u}_i| \leq 2 & \Longrightarrow & \mathbf{y} = \mathbf{z} + (S_G(\mathbf{y}) \,||\, \mathbf{e}_i)
\end{array}$$

De manera que la descodificació s'aconsegueix senzillament calculant el pes de cadascun dels 26 vectors:

$$\begin{array}{c}
S_H,\ S_H + \mathbf{u}_1,\ S_H + \mathbf{u}_2, \ldots,\ S_H + \mathbf{u}_{12}, \\
S_G,\ S_G + \mathbf{u}_1,\ S_G + \mathbf{u}_2, \ldots,\ S_G + \mathbf{u}_{12}
\end{array}$$

on $S_H = S_H(\mathbf{y})$ i $S_G = S_G(\mathbf{y})$.

La proposició 4.9 dóna lloc a l'algorisme 4.1, que és un dels algorismes de descodificació de G_{24}, anomenat *algorisme de Berlekamp*, [Ber68, Ber72], que data del 1968. Tot i que n'existeixen d'altres com "l'algorisme de Kasami" [LC83] o el "descodificador de cerca sistemàtica" [LC83], l'algorisme presentat és el que s'ha implementat a SAGE.

Observant l'algorisme, es veu que aquest, agilitza el procediment genèric, essent només necessari calcular les dues síndromes ($S_H(\mathbf{z})$ i $S_G(\mathbf{z})$) i estalviant el càlcul de la taula de cerca de síndromes, la qual suposaria llistar 4096 files líder–síndrome i suposant això el producte d'un vector (el líder) per una matriu (matriu de control) per a cadascuna de les files.

L'algorisme de correcció de G_{24} s'adapta fàcilment per corregir G_{23}. L'algorisme 4.2 descriu aquest procediment.

Algorisme 4.1 Correcció del codi de Golay binari estès G_{24}

Require: La matriu A i una paraula $\mathbf{z} \in \mathbb{F}_2^{24}$.
Ensure: L'única paraula $\mathbf{y} \in G_{24}$ a distància ≤ 3 de $\mathbf{z}$, si existeix; altrament, *.

for $i \in [1..12]$ **do**
    $\mathbf{u}_i$ = i-èsima fila de A
    $\mathbf{e}_i$ = i-èsima fila de I_{12}
end for

Calcular la síndrome $S_H(\mathbf{z})$.
if $|S_H(\mathbf{z})| \leq 3$ **then**
    $\mathbf{e} = (\mathbf{0} \,||\, S_H(\mathbf{z}))$
    return $\mathbf{y} = \mathbf{z} + \mathbf{e}$
end if

i=1
while $i \leq 12$ **do**
    $\mathbf{x} = S_H(\mathbf{z}) + \mathbf{u}_i$
    if $0 \leq |\mathbf{x}| \leq 2$ **then**
        $\mathbf{e} = (\mathbf{e}_i \,||\, \mathbf{x})$
        return $\mathbf{y} = \mathbf{z} + \mathbf{e}$
    end if
    $i = i + 1$
end while

Calcular la síndrome $S_G(\mathbf{z})$.
if $|S_G(\mathbf{z})| \leq 3$ **then**
    $\mathbf{e} = (S_G(\mathbf{z}) \,||\, \mathbf{0})$
    return $\mathbf{y} = \mathbf{z} + \mathbf{e}$
end if

i=1
while $i \leq 12$ **do**
    $\mathbf{x} = S_G(\mathbf{z}) + \mathbf{u}_i$
    if $0 \leq |\mathbf{x}| \leq 2$ **then**
        $\mathbf{e} = (\mathbf{x} \,||\, \mathbf{e}_i)$
        return $\mathbf{y} = \mathbf{z} + \mathbf{e}$
    end if
    $i = i + 1$
end while

return *

Algorisme 4.2 Correcció del codi de Golay binari G_{23}

Require: La matriu A i una paraula $\mathbf{z} \in \mathbb{F}_2^{23}$
Ensure: L'única paraula $\mathbf{y} \in G_{23}$ a distància ≤ 3 de $\mathbf{z}$

if $|\mathbf{z}|$ és senar **then**
  $a = 0$
else
  $a = 1$
end if

$\hat{\mathbf{z}} = (\mathbf{z} \,||\, a) \in \mathbb{F}_2^{24}$
$\hat{\mathbf{y}}$ = sortida de l'algorisme 4.1 aplicat a $\hat{\mathbf{z}}$
$\mathbf{y}$ = paraula obtinguda a l'eliminar l'últim bit de $\hat{\mathbf{y}}$.
return $\mathbf{y}$

Veiem que l'algorisme 4.1 pot finalitzar retornant el símbol '$*$' $\notin \mathbb{F}_q^k$, que significa que la paraula rebuda no es pot descodificar (conté més de 3 errors). Notar que l'algorisme 4.2 proporciona un esquema de decisió complet ja que al ser G_{23} un codi perfecte sempre descodifica; el resultat mai serà '$*$'.

En els següents capítols es veurà la implementació que s'ha realitzat dels dos codis binaris de Golay en SAGE: un paquet matemàtic de codi obert i lliure distribució.

Capítol 5

Introducció a SAGE

Aquest capítol precedeix al capítol d'implementació i pretén donar al lector una breu introducció al paquet de programari SAGE que ja s'ha presentat breument en el capítol 1.

Sage és un paquet de programari matemàtic de codi obert i lliure distribució que dóna suport a la recerca i la docència en àlgebra, geometria, teoria de nombres, criptografia, computació numèrica i altres àmbits relacionats. Tant el model de desenvolupament de Sage, com Sage en si mateix, es distingeixen per un fort èmfasi en el concepte d'≪Open Source≫, de comunitat, i de cooperació i col·laboració.

L'objectiu de Sage és crear una alternativa de viable, lliure i oberta a Maple, Mathematica, Magma i Matlab; i la seva filosofia és la de "nosaltres estem construint el cotxe, no re-inventant la roda". Aquest objectiu es centra en crear el millor programari per:

- la teoria de nombres ("Quin és el deumilionèsim nombre primer?"),
- l'àlgebra ("Quantes posicions legals té el cub de Rubik?"),
- la geometria ("Quina és l'equació algebraica que descriu la intersecció entre una esfera i un con?"),
- la computació numèrica ("Quin és el deumilionèsim dígit del nombre π?"),
- la probabilitat i l'estadística;

emprant el millor programari lliure ja existent. Així doncs, Sage incorpora actualment: **Maxima** (pel càlcul simbòlic), **Singular** (per l'àlgebra), **R** (per l'estadística), **Pari** (per la teoria de nombres), **SciPy** (per la computació numèrica), **GAP** (per la teoria de grups i de codis) i més d'una seixantena de paquets més[1].

[1] http://www.sagemath.org/links-components.html

Sage va néixer de la mà de William Stein, professor de la Universitat de Washington, Seattle, USA i és liderat pel mateix. La primera versió fou llançada el 24 de febrer de 2005 sota els termes de la GNU GPL ≪General Public License≫ i actualment va per la versió 4.3. La filosofia de desenvolupament de Sage és ≪Release early, release often≫, és a dir, alliberar versions de curta durada i molt sovint per tal de crear una realimentació entre els desenvolupadors i els testejadors del programa.

Quan Stein va pensar en el disseny de Sage, va tenir en compte que per crear una alternativa al programari propietari (Matlab, Maple, Matematica,...) ja existien diferents paquets de codi lliure ben testejats, tot i que estaven escrits en diferents llenguatges. De manera que en lloc de començar des de zero, es va escriure Sage en Python i Cython integrant tot el programari lliure ja existent en una interfície comuna i utilitzant Python com a base. Així, quan no existeixi una alternativa lliure a un problema determinat, aquest serà escrit en Sage però sense la necessitat de reinventar la roda. Així doncs Sage està orientat a la simplicitat i al lliure coneixement.

Però per què Sage? i per què una alternativa lliure? Veiem les opinions d'alguns experts per fer-nos reflexionar sobre aquestes qüestions:

> *"Tu pots llegir un teorema i la seva demostració en algun llibre en una biblioteca [...] després tu pots usar aquest teorema la resta de la teva vida lliure de càrrecs, però en diversos sistemes computacionals d'àlgebra s'han de pagar llicències regularment [...]. Tu prems botons i obtens respostes de la mateixa manera en que veus les imatges brillants del teu televisor però no pots controlar com estan construïdes en cap dels casos.*
>
> *En aquesta situació,* dos de les regles més bàsiques de conducta en les matemàtiques són violades: *En matemàtiques la* informació *és transmesa* lliure de càrrecs *i* tot està exposat a ser comprovat. *La no–aplicació d'aquestes regles en els sistemes computacionals algebraics destinats a la recerca matemàtica [...] significa* moure's en la direcció més indesitjable. *Més important: Podem nosaltres esperar que algú cregui el resultat d'un programa que no li està permès de veure?"*
>
> Joachim Neubüser (1993)
> "pare" de GAP el 1986

> *"Crec, fonamentalment, que el codi obert tendeix a ser programari més estable. És la forma correcta de fer les coses. Jo ho comparo amb la ciència envers la bruixeria. En ciència, tots els sistemes són construïts sobre gent observant els resultats d'altra gent i construint a sobre d'ells. En la bruixeria, algú tenia algun petit secret i el guardava —però mai es permetia als altres realment entendre-ho i profunditzar-hi.*
>
> *El programari tradicional és com la bruixeria. En la història la bruixeria ja ha desaparegut. El mateix succeirà amb el programari. Quan els problemes siguin prou seriosos, no veuràs una persona o una companyia*

guardant els seus secrets. Tindràs que tenir tothom compartint el coneixement."

Linus Torvalds (2004)
El "pare" de Linux.

Així doncs al igual que un matemàtic adquireix el coneixement a través de la lectura d'un teorema o d'una demostració, la gent que treballa en càlculs i algorismes, hauria de poder entendre com es realitzen aquests càlculs sent capaços de llegir un codi font auto–documentat; cosa que no és possible amb la majoria de programes comercials. En Sage però, això no succeeix i qualsevol usuari pot, de forma natural i senzilla, veure el codi de les accions que està executant i aprendre a través del mateix.

Històricament el cicle de vida del programari matemàtic com Matlab, Maple, Mathematica i Magma entre d'altres, ha estat el següent:

- Un projecte de recerca acadèmic (moltes vegades subvencionat per organismes públics),
- fundació d'una companyia comercial,
- obtenció d'immenses quantitats de diners de les universitats i els estudiants (desenes de milions per any!).

En conclusió, la retroalimentació de la informació acadèmica es trenca i el flux d'innovació és canalitzat a través d'una companyia comercial. Entès això, queda més que justificada una iniciativa com aquesta i ajuda a comprendre l'èxit que està tenint entre la comunitat docent.

Sage és desenvolupat per voluntaris i actualment més de 150 persones han contribuït directament en el codi. El desenvolupament de Sage es reparteix en tres llocs:

- una llista de correu,
- el canal d'IRC `#sage-devel` al servidor `irc.freenode.net`,
- un sistema de seguiment (Trac[2]) per al desenvolupament de projectes de programari allotjat en una Web que és on va a parar tot el codi.

Així qualsevol pot col·laborar adaptant codi, generant un pegat i pujant-lo al Trac, on posteriorment és revisat; a banda de discutir o proposar idees a través de la llista de coreu o el canal de xat.

Sage disposa d'una senzilla interfície d'usuari a través de la línia de comandes mitjançant IPython, però disposa també d'una interfície gràfica accessible a través d'un

[2]Més informació sobre Trac a `http://trac.edgewall.org/`

navegador Web anomenada **Notebook**, i no només això sinó que existeix una versió en línia[3] d'aquest Notebook, de manera que qualsevol que disposi d'una connexió a Internet hi pot accedir directament sense haver de realitzar cap mena d'instal·lació.

La interfície gràfica s'estructura en fulls de treball anomenats "worksheets" i permet la compartició dels fulls de treball de diverses formes incloent el correu electrònic o la publicació si es treballa en un servidor públic de Sage o bé si el servidor està disponible en una xarxa local per exemple.

Altres característiques importants a tenir en compte són:

- suport al processament en paral·lel, ja sigui a través d'un processador multi–nucli o de la computació distribuïda,
- versions per a diferents sistemes operatius (GNU/Linux, Mac OS X, Solaris, portació nativa per MS Windows en procés),
- processador de textos tècnics, incloent l'edició e formules amb LaTeX, així com la inversa: inclusió de codi, resultats de càlculs i gràfics en documents de LaTeX mitjançant el paquet `sagetex`.
- gràfics en 2D i 3D,
- i un llarg etcètera.

En quant al rendiment en temps de còmput, alguns resultats de diversos "Benchmarks" es troben disponibles al Web de Sage[4] i que mostren com Sage obté millors resultats en moltes de les proves que altres paquets "rivals".

[3] http://www.sagenb.org
[4] http://www.sagemath.org/tour-benchmarks.html

Capítol 6

Implementació en SAGE

Ja s'ha vist en el capítol anterior que Sage està en constant desenvolupament i que no totes les funcionalitats estan implementades de forma nativa. Concretament en referència a la teoria de codis, Sage depèn en bona mesura del paquet GAP/GUAVA inclòs en el mateix. Tot i que, part del —per altra banda escàs— codi, està implementat de forma nativa. Aquest capítol descriu la implementació que s'ha realitzat dels codis binaris de Golay presentats en el capítol 4.

6.1 Antecedents

6.1.1 Codis lineals i construccions

Sage implementa els codis lineals des del novembre de 2005. David Joyner, professor del departament de matemàtiques de la U. S. Naval Academy, Annapolis, USA, i coautor del paquet Guava per a teoria de codis de GAP, juntament amb William Stein —fundador de Sage— és qui ha escrit pràcticament la totalitat del codi referent a teoria de codis. Tot i això els codis lineals estan implementats de forma genèrica. Sí existeixen però les construccions de gran varietat de codis, però només la construcció.

Així doncs resulta evident i necessari implementar de forma específica la major quantitat de codis coneguts possibles per tal d'aprofitar-ne les propietats característiques de cadascun. Per exemple, tal i com s'ha vist en la secció 3.5, la descodificació de codis grans mitjançant el mecanisme genèric per als codis lineals és poc eficient.

Les construccions dels codis de Golay (incloent els dos ternaris) ja existien en Sage, però s'han modificat les matrius generadores dels dos codis de Golay binaris per les matrius presentades al capítol 4 que estan més estandarditzades. Les matrius que implementava Sage provenen —tal i com m'ha reportat el mateix Joyner— de

Guava i no s'acostumen a trobar en la literatura de codis correctors d'errors. Cal assenyalar que GAP, que és d'origen alemany, utilitza per als cossos finits notacions difícils de seguir i poc estàndards, de manera que s'han modificat aquestes matrius amb el vist–i–plau de D. Joyner.

La implementació de les construccions dels codis i dels codis lineals, es troba en els següents mòduls Python de Sage:

- `sage.coding.code_constructions.py`
- `sage.coding.linear_code.py`

També s'ha realitzat un petit canvi en el mètode `dual_code` de la classe `LinearCode` del mòdul dels codis lineals, encarregat d'obtenir el codi dual (i que és cridat també a l'hora d'obtenir la matriu de control d'un codi); incloent-hi el paràmetre opcional `"method"`. Aquest paràmetre permet obtenir la matriu de control d'un codi en la forma sistemàtica ($H = \left(A^T \mid I_r\right)$) quan `method="systematic"`, el qual pot resultar interessant per obtenir la matriu de control o construir el codi dual de diferents formes. Un segon "`method`" i que ja estava "mig" implementat (no funcionava) i comentat en el codi original, també s'ha inclòs. Aquest s'ha anomenat `"sage2"` i obté la matriu de control com $H = \left(I_r \mid A^T\right)$. La crida sense paràmetre o `method="default"`, obté la matriu de control com $H = \left(I_r \mid B\right)$.

Aquest paràmetre també s'ha inclòs en el mètode `check_mat` de la mateixa classe per tal d'obtenir la matriu de control, tal com s'ha comentat, en diferents formes.

6.1.2 Descodificació

Sage incorpora també un mòdul per tal de descodificar els codis lineals, implementat per D. Joyner al febrer de 2009. Aquest descodificador però, proveeix d'un mecanisme estàndard per als codis Lineals basat en els conceptes de *mínima distància* i *taula d'Slepian* descrits en la secció 3.5. Aquests mecanismes, a banda de ser força ineficients per als codis grans —com és el cas dels Golay—, són esquemes de decisió complets, es a dir, no proveeixen cap mecanisme per a la no–descodificació quan aquesta sigui ambigua. La implementació d'aquest mòdul es troba en el mòdul Python de Sage `sage.coding.decoder.py`.

El mètode `decode` de la classe `LinearCode` del mòdul dels codis lineals, admet un paràmetre en la seva crida que fa que es realitzi la descodificació a través de GAP, el qual millora substancialment el temps de descodificació al emprar (suposadament) l'algorisme de descodificació via síndrome descrit a la secció 3.5.2. Però tot i això el temps de descodificació pot ser millorat aplicant un algorisme específic, i igualment GAP/Guava es basa un esquema de descodificació complet donant lloc a possibles descodificacions errònies.

La figura 6.1 mostra una captura de la sortida de línia de comandes de Sage i posa de manifest aquests fets. Com es pot observar en la mateixa figura, al intentar corregir una paraula amb 4 errors, els dos mètodes (Sage i GAP/Guava) retornen una paraula–codi incorrecta. També es veu com la crida a GAP millora el temps de descodificació.

```
sage: G24 = ExtendedBinaryGolayCode(); G24
Linear code of length 24, dimension 12 over Finite Field of size 2
sage: y
(1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0)
sage: z
(1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0)
sage: y in G24
True
sage: z in G24
False
sage: hamming_weight(z-y)
4
sage: time v=G24.decode(z)
CPU times: user 3.25 s, sys: 0.00 s, total: 3.25 s
Wall time: 3.76 s
sage: v==y
False
sage: time v=G24.decode(z,"guava")
CPU times: user 0.23 s, sys: 0.01 s, total: 0.24 s
Wall time: 1.00 s
sage: v==y
False
```

Figura 6.1: Sortida de la línia de comandes de Sage

El principal problema que suposa la descodificació mitjançant GAP, és un retard addicional pel fet de realitzar una crida externa. Tal i com es veu a l'anterior figura, el temps de resposta és gairebé cinc vegades el temps de còmput. Tot i això s'obtenen temps totals millors que amb el descodificador natiu.

6.2 Implementació dels codis de Golay

Per tal de suplir les mancances relatives a la descodificació descrites anteriorment, s'ha implementat en un nou mòdul la classe `GolayCode` com a extensió de `LinearCode` per tal de redefinir el mètode `decode`. La implementació es basa en els algorismes 4.1 i 4.2. S'han realitzat dues petites modificacions sobre l'algorisme base (4.1):

- s'han ajuntat els dos bucles que recorren les files de les matrius A i I_{12}, deixant

el bucle resultant al final de l'algorisme per tal d'agilitzar el procés de descodificació en aquells casos on no sigui necessari fer cap recorregut;

- es contempla un cas on $|S_H(\mathbf{z})|$ o $|S_G(\mathbf{z})|$ és 4, pel qual es determina que no es pot descodificar i s'ha de demanar retransmissió, que evita entrar en el citat bucle.

El codi de la implementació es troba en el mòdul Python de Sage creat per a tal efecte, `sage.coding.binary_golay_code.py`.

A banda de redefinir el mètode de descodificació (`decode`), també ha estat necessari redefinir aquells mètodes heretats de la classe base `LinearCode` que retornaven un codi lineal per a fer que retornessin un codi de Golay en els casos on la crida al mètode no feia que el codi deixés de ser un codi de Golay. Aquests són els mètodes:

- `extended_code`, quan es crida sobre el codi G_{23}, donant lloc al codi G_{24};
- `punctured`, quan s'aplica sobre la darrera posició del codi G_{24}, donant lloc al codi G_{23};
- `standard_form`, que retorna una tupla amb un codi equivalent al codi sobre el qual es crida, amb matriu generadora en forma sistemàtica[1] i la permutació realitzada;
- `dual_code`, que obté el codi dual al codi sobre els que es crida (notar que G_{24} és autodual, per tant s'obté el mateix codi);
- `permuted_code`, que retorna un codi equivalent generat per una permutació de les columnes del codi sobre el que es crida.

També s'ha implementat una petita funció (`syndrome(C,v,method)`), necessària en l'algorisme de descodificació, que calcula la síndrome d'una paraula, donat un codi i la paraula. Aquesta funció, que rep un codi lineal, un vector o paraula i opcionalment el mètode per calcular la matriu de control; s'ha escrit al mòdul dels codis lineals i s'ha exportat per tal que pugui ésser utilitzada per l'usuari de Sage amb qualsevol codi.

Amb les modificacions realitzades, s'ha realitzat un pegat del codi i s'ha enviat al servidor Trac de Sage amb la finalitat que sigui revisat per la resta de desenvolupadors i sigui inclòs en en pròximes versions de Sage si és acceptat.

Tot el codi implementat, s'ha comentat degudament seguint els convenis de Sage amb les descripcions dels mètodes, classes i funcions escrites. Els fitxers de codi font que intervenen en la implementació, s'han adjuntat al CD-ROM que acompanya

[1] Les construccions dels codis de Golay ja es fan mitjançant matrius generadores en forma sistemàtica.

aquest treball, així com un pegat amb els canvis realitzats. Aquest pegat no és més que un fitxer `.patch` que conté les diferències entre els fitxers originals i les modificacions, i que pot ser inspeccionat amb un editor de text.

Davant els dubtes que han anat sorgint de cara a la implementació, tant el canal d'IRC de Sage com la llista de correu sage-devel, han estat d'ajuda per a resoldre'ls. La possibilitat de contactar de forma directa amb els mantenidors i desenvolupadors sempre és de gran ajuda i aquesta és una de les característiques de les comunitats de programari lliure. En l'apèndix A es recullen algunes d'aquestes comunicacions via correu electrònic.

6.3 Proves i resultats

Per tal de veure el resultat d'una primera prova de la implementació, es reprodueix l'exemple de la figura 6.1 emprant aquest cop el descodificador específic de la nova classe implementada per als codis de Golay. La figura 6.2 mostra la sortida de la línia de comandes de Sage. Fixem-nos que aquest cop la paraula **z** que conté quatre errors no és descodificada i se'n demana la retransmissió i fixem-nos també en el temps que s'ha pres aquesta determinació.

```
sage: G24=ExtendedBinaryGolayCode(); G24
Binary Golay code G24
sage: y
(0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0)
sage: z
(0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0)
sage: y in G24
True
sage: z in G24
False
sage: hamming_weight(z-y)
4
sage: time v=G24.decode(z)
CPU times: user 0.08 s, sys: 0.00 s, total: 0.08 s
Wall time: 0.11 s
sage: v
'*'
```

Figura 6.2: Sortida de la línia de comandes de Sage

En les següents proves es contrastarà la diferència de temps en descodificació dels diferents mètodes (mètodes ja existents i nova implementació), es a dir, es provarà el rendiment en descodificació per a la capacitat correctora dels codis de Golay.

Per tal de provar el rendiment de forma més exhaustiva, s'han escrit dues petites

funcions directament en un full de treball de Sage per tal de mesurar els temps de descodificació i calcular-ne la mitjana:

- `rand_bin_err(v,ne)` retorna la paraula binària **v** amb exactament `ne` errors. Aquesta funció rep una paraula–codi $\mathbf{v} \in \mathbb{F}_2^n$ i un enter $\texttt{ne} \leq n$;

- `golay_test(code,e,n)` realitza un test de descodificació. Aquesta funció crida `rand_bin_err` i recorre totes les paraules–codi de G_{23} o G_{24} (segons el paràmetre `code`), afegeix fins a `e` errors de forma iterativa i les descodifica `n` vegades amb un error diferent cada vegada, calculant-ne la mitjana de temps i els temps màxims i mínims per a 1,...,`e` errors successivament.

La funció `golay_test` s'ha utilitzat també per veure que l'algorisme implementat no falla. Per cada paraula que l'algorisme descodifica, la funció comprova si la paraula corregida és igual a la paraula–codi a la que s'han afegit els errors. En cas de no coincidir, s'incrementa un comptador de fallades. Si aquest comptador es manté sempre a zero, es tenen indicis[2] de que l'algorisme funciona correctament.

Per tal de provar en la implementació dels codis de Golay el mètode de descodificació genèrica dels codis lineals, es fa ús de la funció `super` de Python, que permet cridar al mètode de descodificació de la classe base, es a dir de la classe `LinearCode`. Així es pot contrastar l'eficiència de la implementació.

El full de treball que conté les funcions dels tests s'ha adjuntat també al CD-ROM que acompanya aquest treball. Es tracta d'un fitxer amb extensió `.sws` i que es pot importar al Notebook de Sage.

Tot seguit es mostren els resultats dels tests realitzats. Tots els tests han estat realitzats sobre el codi G_{24}. La figura 6.3 mostra els resultats obtinguts el en full de treball `Golay_test-full`, que realitza el test amb els tres mètodes esmentats: Sage, Guava i la implementació Golay. També es mostra un gràfic de la mitjana de temps requerit per cada mètode. Cal notar que aquests són els temps totals i no els temps de CPU. Com ja s'ha comentat, tot i que la descodificació mitjançant GAP/Guava obté uns temps de CPU bastant bons, el temps total en la descodificació es dispara. Els gràfics que es veuen a continuació mostren una escala logarítmica enlloc de lineal per una millor visualització dels resultats. El temps de realització d'aquest test ha estat ≈ 15 hores, mentre que el temps de CPU només de 8,5 hores.

[2] Per assegurar-ho s'haurien de comprovar les $4096 \cdot \left(\binom{24}{1} + \binom{24}{2} + \binom{24}{3}\right) = 9519104$ paraules diferents, resultants de repartir 1, 2 i 3 errors en totes les possibles posicions de cada paraula–codi.

```
Mitja de temps Golay:  [0.027074521640315652, 0.028260672115720809,
0.02997182693798095]
Min temps Golay:  [0.025668859481811523, 0.026736974716186523,
0.026790142059326172]
Max temps Golay:  [0.081119775772094727, 0.073632955551147461,
0.077738046646118164]
Fallades Golay:  0

Mitja de temps synd:  [2.5061827643075958, 2.5089990932028741,
2.509469363023527]
Min temps synd:  [2.425724983215332, 2.4295589923858643,
2.4298100471496582]
Max temps synd:  [3.1317310333251953, 2.732820987701416,
2.7587738037109375]
Fallades synd:  0

Mitja de temps Guava:  [1.7627230762736872, 1.7758685279986821,
1.7769823423004709]
Min temps Guava:  [1.6499080657958984, 1.6512050628662109,
1.583186149597168]
Max temps Guava:  [2.1339950561523438, 2.1508090496063232,
2.1804499626159668]
Fallades Guava:  0
Time: CPU 30671.56 s, Wall: 52960.68 s
```

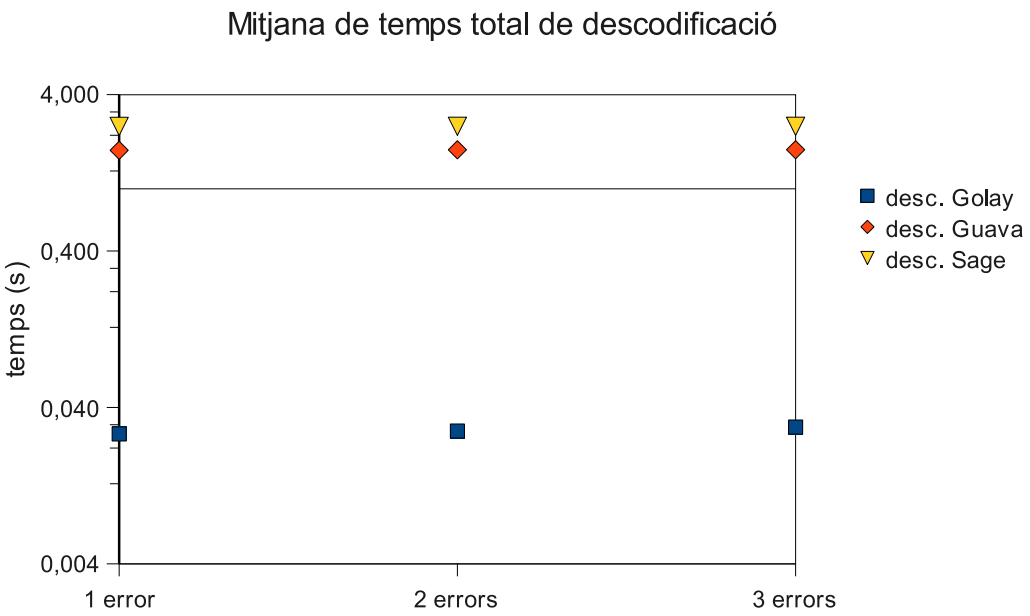

Figura 6.3: Resultats Golay_full-test

A continuació es mostren els resultats de la descodificació individual amb cadascun dels mètodes, obtinguts en el full de treball `Golay_test`, per tal de contrastar el temps de CPU amb el temps total per cadascun dels tres mètodes. La figura 6.3 en mostra un gràfic.

- **Descodificació Golay:**

```
Mitja de temps: [0.027351205470040441, 0.027633182995487005, 0.02929115790175274]
Min temps:  [0.025071144104003906, 0.026340961456298828, 0.026298999786376953]
Max temps:  [0.13699913024902344, 0.1324460506439209, 0.14296698570251465]
Fallades:  0
Time: CPU 346.28 s, Wall: 356.21 s
```

Figura 6.4: Resultat descodificació Golay

- **Descodificació Guava:**

```
Mitja de temps: [1.7607569640967995, 1.7613838939578272, 1.7622313382453285]
Min temps:  [1.5992171764373779, 1.6194381713867188, 1.5529618263244629]
Max temps:  [2.2559530735015869, 1.9590280055999756, 2.1709089279174805]
Fallades:  0
Time: CPU 103.98 s, Wall: 21656.61 s
```

Figura 6.5: Resultat descodificació Guava

- **Descodificació Sage:**

```
Mitja de temps: [2.5619092612760141, 2.5630878108786419, 2.5617000786587596]
Min temps:  [2.4848132133483887, 2.4851338863372803, 2.4858908653259277]
Max temps:  [3.5680241584777832, 2.9287979602813721, 2.8993818759918213]
Fallades:  0
Time: CPU 30889.84 s, Wall: 31498.75 s
```

Figura 6.6: Resultat descodificació Sage

Veiem doncs que la descodificació mitjançant la implementació dels codis de Golay, triga de l'ordre de 10^2 vegades menys.

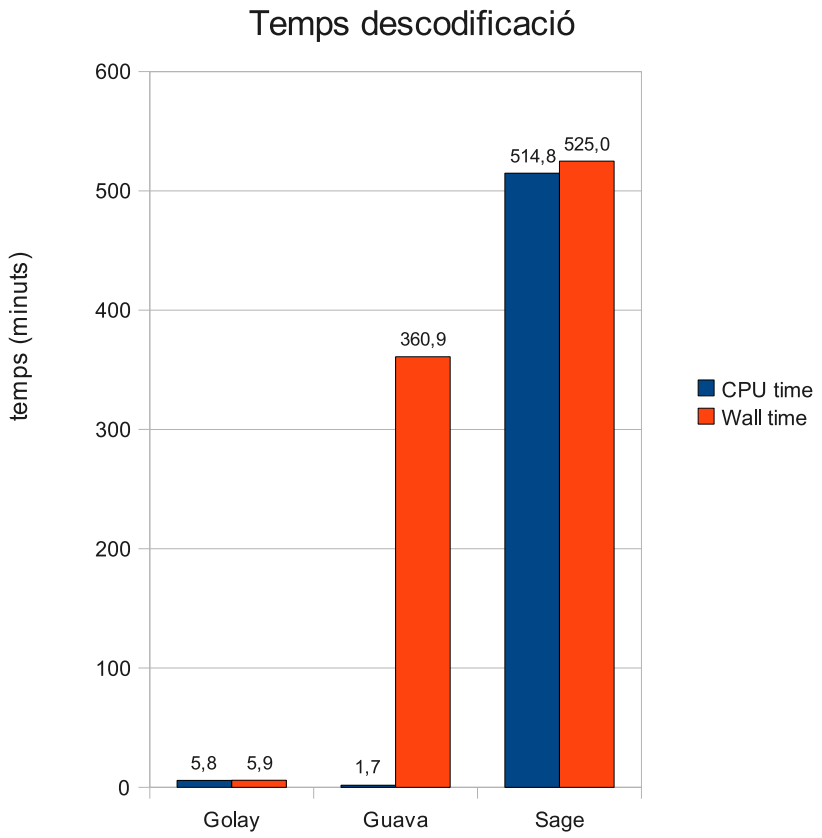

Figura 6.7: Histograma del temps de descodificació

Finalment amb l'objectiu de provar de forma més exhaustiva la implementació, s'ha repetit —mitjançant la funció `golay_test`— la descodificació de cada paraula–codi 10 vegades en el test anterior, afegint un error aleatori diferent en cada iteració a la paraula $\mathbf{y}$. La realització del test ha pres menys d'una hora i només ha estat realitzada amb el descodificador de Golay degut al temps que hauria pres amb els altres dos descodificadors. Aquesta prova, descodifica igualment totes les paraules–codi amb 1, 2 i 3 errors. La següent figura en mostra el resultat.

```
Mitja de temps: [0.02658449605805794, 0.027692925487645018, 0.029279980930732537]
Min temps:  [0.024226188659667969, 0.026272058486938477, 0.026304960250854492]
Max temps:  [0.18466901779174805, 0.18136811256408691, 0.17159390449523926]
Fallades:  0
Time: CPU 3395.27 s, Wall: 3467.51 s
```

Figura 6.8: Descodificació exhaustiva Golay

El maquinari i sistema operatiu utilitzat per la realització de les proves es descriu a continuació:

- Processador: AMD Athlon 64 3000+ (1.8GHz)
- Memòria principal: 2GB RAM DDR2
- Sistema operatiu: GNU/Linux Ubuntu 9.10.
- Sage versió 4.1.2 amb el pegat de codi aplicat.

Capítol 7

Conclusions i treball futur

Abans de realitzar una valoració personal i conclusió d'aquest Treball de Final de Carrera, voldria assenyalar que el treball ha arribat al seu fi havent assolit l'objectiu que va donar peu al mateix: la implementació d'un descodificador específic per als codis de Golay, de forma nativa en el codi de Sage. Es ha dir, s'ha aconseguit fer el que es pretenia i se n'ha pogut avaluar el funcionament. Assenyalo aquest fet, ja que moltes vegades en el transcurs d'un projecte poden ocórrer situacions no previstes o interposar-se entrebancs que en poden fer girar el rumb.

En aquestes pàgines s'ha mostrat la utilitat dels codis de Golay com a codis correctors d'errors, però des que M. J. Golay descobreix aquests codis fins que E. Berlekamp en desenvolupa un algorisme de descodificació, transcorren gairebé vint anys. Això és degut a que Golay va desenvolupar els codis amb finalitats diferents de la correcció d'errors i més purament matemàtiques. El fet d'haver escollit l'algorisme de Berlekamp es deu a que és un dels algorismes clàssics i que a banda de ser senzill, és específic per als codis de Golay, a diferència d'altres algorismes que són vàlids per als codis cíclics en general.

Cal esmentar de la recerca que s'ha realitzat, que sorprèn l'existència d'algorismes molt recents de descodificació per als codis de Golay com [LTS07] o [LWC95] i que són, en alguns casos, capaços de corregir determinats patrons de fins a 4 errors. Això demostra que encara avui dia els codis de Golay segueixen essent objecte d'estudi.

Sembla ser que les vies de treball futur entorn a Sage i entorn no tan sols als codis de Golay, sinó a la teoria de codis en general, estan doncs força obertes. En un futur nous descodificadors podrien ser implementats, així com altres codis, ja que actualment Sage només te implementats els codis de Hamming. També es troba en falta un descodificador via síndrome implementat de forma nativa.

Els resultats obtinguts en les proves realitzades han estat satisfactoris i per tant el codi de la implementació s'ha tramés al servidor de Sage on s'allotja i revisa tot el codi font, amb la finalitat que sigui revisat i inclòs en pròximes versions.

Finalment, en quant a l'aprofitament personal que la realització d'aquest treball m'ha aportat, voldria destacar-ne breument alguns dels aspectes més rellevants com,

- l'estudi d'aspectes tant teòrics com pràctics de la matemàtica i la teoria de codis;
- l'exploració de bibliografia matemàtica i textos en anglès;
- la immersió en l'ús de LaTeXper a la composició de textos tècnics/científics i presentacions;
- la comunicació amb una comunitat de programari lliure;
- la investigació en el funcionament i estructura de Sage;
- la participació en el model de desenvolupament del programari lliure;
- el desenvolupament de codi en Python.

Apèndix A

Contacte amb la comunitat SAGE

Les següents pàgines mostren alguns dels contactes establerts amb la comunitat de desenvolupadors de Sage per tal de resoldre algun dels dubtes sobre el disseny i la implementació que van sorgir.

Gmail by Google

Gerard Bosch <gerard.bosch@gmail.com>

[Sage] Golay codes development

2 messages

Gerard Bosch <gerard.bosch@gmail.com> **23 November 2009 18:16**
To: wdjoyner@gmail.com

Hello,

my name is Gerard and I'm working in an implementation for Sage about decoding methods for the Binary Golay Codes (G24 and G23) for my last degree project at university in Spain of Computer Science.

I'm implementing an algorithm to decode the Binary Golays.

I'm writing you because I have seen you have written the most part of the Coding theory in Sage and in particular for one question about the construction of the G24 and G23:

In all bibliography I have checked, the way to construct the G24 and consequently the G23 (by deleting the last bit of generator matrix G) is trough the generator matrix G=(I_k | A), where 'I_k' is the (k x k) identity matrix and 'A' is the sub-matrix that follows:

```
A  = [[1,1,0,1,1,1,0,0,0,1,0,1],
      [1,0,1,1,1,0,0,0,1,0,1,1],
      [0,1,1,1,0,0,0,1,0,1,1,1],
      [1,1,1,0,0,0,1,0,1,1,0,1],
      [1,1,0,0,0,1,0,1,1,0,1,1],
      [1,0,0,0,1,0,1,1,0,1,1,1],
      [0,0,0,1,0,1,1,0,1,1,1,1],
      [0,0,1,0,1,1,0,1,1,1,0,1],
      [0,1,0,1,1,0,1,1,1,0,0,1],
      [1,0,1,1,0,1,1,1,0,0,0,1],
      [0,1,1,0,1,1,1,0,0,0,1,1],
      [1,1,1,1,1,1,1,1,1,1,1,0]]
```

that is symmetric and satisfy the Antiortogonal proposition (A*A^t = I_k); and the sub-matrix that results to remove the last column and row from 'A' is a circulant matrix (i.e. each row is obtained from the previous by shifting their components).

But the matrix that Sage is using for the Golay codes is very different:

Sage Generator for Golay24:

```
B = [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1],\
      [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
0, 0, 1, 0],\
      [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 0, 1, 1],\
      [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1,
0, 1, 1, 0],\
      [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
1, 0, 0, 1],\
```

```
        [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
1, 1, 0, 1],\
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
0, 1, 1, 1],\
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
1, 0, 0, 0],\
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
1, 1, 0, 0],\
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1,
1, 1, 1, 0],\
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,
1, 1, 0, 1],\
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
0, 1, 1, 1]]
```

where sub-matrix A is very diferent? Have this matrix any special feature or something?
Could you give me some reference where I can know about or simply tell me why this matrix is different?

I'm only learning for the Golay codes, that's the aim of the project and I'm new with Sage and sage-devel, of course. If you have any suggestion I'd be grateful to know. I think that make a new python class for the Golay codes with his own decode method would be the thing...

Thanks in advance,
Gerard.

David Joyner <wdjoyner@gmail.com> **23 November 2009 19:09**
To: Gerard Bosch <gerard.bosch@gmail.com>

First, thanks for working on this - the more coding theory developers Sage has, the better!

I think (but I am not sure) that the generator matrix for the Golay codes for Sage are the same as those for Guava (a GAP package for error-correcting codes). I don't know if there is any special structure though. I think the Sage version is in "standard form", whereas the Guava version might not be (I don't remember) but after doing a row-reduction, I think they are the same. Your version is fine though and can be used as the default if that is useful, and if you create the Sage patch to make the change.

Adding a new GolayCodes class with a specialized decoding method would be great too! Please make a patch for that as well.

[Quoted text hidden]

Gmail by Google

Gerard Bosch <gerard.bosch@gmail.com>

Implementing error-correction Golay codes

4 messages

Gerard Bosch <gerard.bosch@gmail.com> **12 December 2009 16:35**
To: sage-devel@googlegroups.com

Hello,

First of all I have to say that I'm newbie with Sage. And I read that it's a good idea to start a disscusion here in order to implement new functionalities in Sage.

I'm working with the implementation of the Golay codes, in particular for the binary Golays (G23 and G24). I have made a new python module for this purpose with an specific decoder method. The algorithm I have implemented is based in the weight of the syndromes of the codewords to decode. The algorithm is an incomplete decoding scheme (i.e. can correct a number of errors <= correcting capacity (t) of the code; 3 in the case of Golays). So is capable to correct up to 3 errors (and detect up to 4 errors simultaneously for G24).

I already have implemented it and works fine and quick to decode up to 3 errors. But now there is only a problem and it's that I don't know how to treat the no-decoding cases (i.e. the cases in G24 which the word to decode contains 4 errors). Now the method returns a vector with the corrected word (if can decode) or the string "*" if more than 3 errors was found (only for G24; G23 is perfect).

I think it's a problem that a function that should return a codeword could return a string ("*") to comunicate that decoding is not possible, so I would like to know how should I treat this cases. ¿...raising an exception or something like this?, how the error cases are usually treated in Sage?

I hope for your replies and any advice that could help me with this question or related to devel process (steps to follow, etc...)

Thanks,
Gerard Bosch.

Gerard Bosch <gerard.bosch@gmail.com> **12 December 2009 16:38**
To: sage-devel@googlegroups.com

P.S. In fact "*" should means to ask for a retransmission of the word.

Gerard Bosch.

2009/12/12 Gerard Bosch <gerard.bosch@gmail.com>:
[Quoted text hidden]

David Joyner <wdjoyner@gmail.com> **12 December 2009 17:56**
Reply-To: sage-devel@googlegroups.com
To: sage-devel@googlegroups.com

On Sat, Dec 12, 2009 at 10:35 AM, Gerard Bosch <gerard.bosch@gmail.com> wrote:
> Hello,
>
> First of all I have to say that I'm newbie with Sage. And I read that
> it's a good idea to start a disscusion here in order to implement new
> functionalities in Sage.
>
> I'm working with the implementation of the Golay codes, in particular
> for the binary Golays (G23 and G24). I have made a new python module
> for this purpose with an specific decoder method. The algorithm I have
> implemented is based in the weight of the syndromes of the codewords
> to decode. The algorithm is an incomplete decoding scheme (i.e. can

Syndrome decoding. Is your procedure faster than the syndrome
decoding already implemented in Saeg (see the sage/coding/decoder.py
module for details)?

> correct a number of errors <= correcting capacity (t) of the code; 3
> in the case of Golays). So is capable to correct up to 3 errors (and
> detect up to 4 errors simultaneously for G24).
>
> I already have implemented it and works fine and quick to decode up to
> 3 errors. But now there is only a problem and it's that I don't know
> how to treat the no-decoding cases (i.e. the cases in G24 which the
> word to decode contains 4 errors). Now the method returns a vector
> with the corrected word (if can decode) or the string "*" if more than
> 3 errors was found (only for G24; G23 is perfect).
>
> I think it's a problem that a function that should return a codeword
> could return a string ("*") to comunicate that decoding is not
> possible, so I would like to know how should I treat this cases.
> ¿...raising an exception or something like this?, how the error cases
> are usually treated in Sage?

My opinion is to either return

(a) an arbitrarily selected (but deterministic) closest codeword,
(b) revert to list decoding,
(c) both - namely return by default a closest codeword, but allow
a keyword option which returns the list of all closest codewords.

Maybe others have a different opinion though.

Thanks for working on this!!

>
>
> I hope for your replies and any advice that could help me with this
> question or related to devel process (steps to follow, etc...)
>

> Thanks,
> Gerard Bosch.
>
> --
> To post to this group, send an email to sage-devel@googlegroups.com
> To unsubscribe from this group, send an email to sage-devel+unsubscribe@googlegroups.com
> For more options, visit this group at http://groups.google.com/group/sage-devel
> URL: http://www.sagemath.org
>

--
To post to this group, send an email to sage-devel@googlegroups.com
To unsubscribe from this group, send an email to sage-devel+unsubscribe@googlegroups.com
For more options, visit this group at http://groups.google.com/group/sage-devel
URL: http://www.sagemath.org

Gerard Bosch <gerard.bosch@gmail.com> **12 December 2009 21:16**
To: sage-devel@googlegroups.com

The procedure I have implemented works faster because is an specific decoder for the Golay codes and it's based on the syndrome computation (s(v) = v*H^t, where H is the check_mat). I think that the syndrome implemented in (coding/decoder.py) it's not a real syndrome decoding. It consists in list the whole coset v+C and sort it by weight, and it's a complete decoding scheme so it can correct erroneously words with more than "t" errors when the minimum distance don't correspond to a unique codeword. The size of G24 is of 4096 codewords and takes a little time to list and sort.

The procedure I have implemented is based only on the weigth of the syndrome s(v) and works really fast between ~0.04 and ~0.4 seconds in my old computer.

I'm doing this for my last degree project in Spain.

I'll think what can I do with the 4-error words, but I'd like something like ask retransmission or any warning.

Thanks,
Gerard Bosch.

2009/12/12 David Joyner <wdjoyner@gmail.com>:
[Quoted text hidden]

Bibliografia

[Ber68] Elwyn R. Berlekamp, *Algebraic Coding Theory*, revised 1984 ed., Aegean Park Press, 1968.

[Ber72] E. R. Berlekamp, *Decoding the Golay code*, JPL Technical Report 32-1526 **XI** (1972), 81–85.

[BV01] J. M. Brunat Blay and E. Ventura Capell, *Informació i codis*, Politext, no. 114, Edicions UPC, 2001.

[CHP03] W. Cary Huffman and V. Pless, *Fundamentals of error–correcting codes*, Cambridge University Press, 2003.

[Hil93] Raymond Hill, *A First Course in Coding Theory*, Oxford Applied Mathematics and Computing Science Series, Clarendon Press, 1993.

[LC83] Shu Lin and Daniel J. Costello, *Error Control Coding: Fundamentals and Applications*, Pentice Hall, 1983.

[LTS07] T. C. Lin and T. K. Truong and W. K. Su and P. Y. Shih and G. Dubney, *Decoding of the (24,12,8) extended Golay code up to four errors*, Institute of Engineering and Technology Commun., **3** (2009), lss. 2, 232–238.

[LWC95] E. H. Lu and H. P. Wuu and Y. C. Cheng and P. C. Lu, *Fast algorithm for decoding the (23,12,7) binary Golay code with four–error–correcting capability*, Int. J. Syst. Sci., **26** (1995), (4), 93–945.